UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Rafael Besen

DESCOBERTA SEMÂNTICA DE SERVIÇOS EM AMBIENTES COM DISPOSITIVOS MÓVEIS

Dissertação submetida ao Programa de Pós-Graduação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciências da Computação
Orientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis

2011

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

B554d Besen, Rafael
Descoberta semântica de serviços em ambientes com dispositivos móveis [dissertação] / Rafael Besen ; orientador, Frank Augusto Siqueira. - Florianópolis, SC, 2011.
87 p.: il., grafs., tabs.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da computação. 2. Serviços da Web. 3. Semântica. 4. Ontologia. I. Siqueira, Frank. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 681

Rafael Besen

DESCOBERTA SEMÂNTICA DE SERVIÇOS EM AMBIENTES COM DISPOSITIVOS MÓVEIS

Esta Dissertação foi julgada adequada para obtenção do Título de "mestre",e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Santa Catarina

Florianópolis, 31 de agosto de 2011.

_____
Prof. Mário Antonio Ribeiro Dantas, Ph.D.
Coordenador do Curso

Banca Examinadora:

_____
Prof. Mário Antonio Ribeiro Dantas, Ph.D.
Universidade Federal de Santa Catarina

_____
Prof., Lau Cheuk Lung, Dr.
Universidade Federal de Santa Catarina

_____
Prof., Claudio Renato Resin Geyer, Ph.D.
Universidade Federal do Rio Grande do Sul

Dedico este trabalho a todos familiares e amigos, que apoiaram o desenvolvimento deste trabalho, demonstrando paciência e carinho. E a meus colegas de trabalho que foram compreensivos nos momentos difíceis. Demonstrando que a superação nesses momentos vale a pena.

AGRADECIMENTOS

Agradeço ao Professor Dr. Frank Augusto Siqueira, pela orientação e constante incentivo para desenvolvimento desta dissertação. Sempre disponível para trabalhar em novas idéias e indicar caminhos a serem seguidos.

Agradeço a todos os professores do PPGCC, que no inicio do trabalho proporcionaram além de uma forte base para o desenvolvimento desta dissertação, se apresentaram sempre disponíveis para sanar dúvidas, colaborando diretamente para a conclusão deste trabalho.

À SIMB Tecnologia por prover não apenas a infraestrutura necessária, mas todo o incentivo para a conclusão desta dissertação.

"Antes que você possa alcançar o topo de uma árvore e entender os brotos e as flores, você terá de ir fundo nas raízes, porque o segredo está lá. E, quanto mais fundo vão as raízes, mais alto vai a árvore"

(Nietzsche, 1883)

RESUMO

Este trabalho de mestrado apresenta um modelo de descoberta semântica de serviços, focando principalmente em serviços providos e acessados por dispositivos móveis. O modelo proposto visa aprimorar a tecnologia dos modelos de descoberta de serviços atuais, os quais em sua maioria baseiam-se em descrições sintáticas dos serviços. Além disso, temos a preocupação de manter uma arquitetura que utilize padrões de mercado e seja leve o suficiente para executar em dispositivos com recursos limitados. Para atingir esses objetivos, utilizamos Serviços Web Semânticos. Os serviços são descritos através de ontologias criadas a partir do WSMO, e para realizar o *matching* semântico utilizamos o ambiente de execução WSMX. Para que a descoberta e a invocação dos serviços sejam viáveis em ambientes com recursos limitados, utilizamos o padrão recomendado pelo OASIS, o DPWS. Foi considerado um fator importante neste trabalho o DPWS suportar descoberta semântica, porém se mantendo compatível com componentes já existentes em ambientes sintáticos. Houve a necessidade do desenvolvimento de um componente para integrar as partes da arquitetura, viabilizando o modelo. A principal contribuição deixada pela dissertação é um modelo que possibilita a descoberta semântica e invocação automática de serviços em um ambiente distribuído, serviços esses que podem ser acessados e providos tanto por dispositivos móveis com recursos limitados quanto estações fixas com mais recursos.

Palavras-chave: Serviços Web. Descoberta Semântica. Ontologia. WSMO.

ABSTRACT

This paper presents a model for semantic Service discovery, which focuses mainly on Services provided and accessed by mobile devices. The model aims to improve the current Service discovery technology, which is mainly based on syntactic descriptions of Services. Moreover, we are concerned in maintaining an architecture that uses market patterns and is lightweight enough to run on devices with limited resources. To achieve these goals, we use semantic Web Services. Services are described through ontologies created with WSMO. We use the runtime WSMX to execute the semantic matching and the DPWS standard to enable Service discovery and invocation in an environment with limited resources. The DPWS standard is recommended by OASIS, and was necessary to extend it to support semantic description and to continue compatible with components already built. Moreover, a component that integrates all parts of architecture has been developed. The main contribution resulting from this work is a model that enables automatic semantic Service discovery and invocation in a distributed environment. Services can be accessed and provided either by mobile devices with limited resources or by workstations in a fixed network with more resources.

Keywords: Semantic Discovery. Web Services. WSMO.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface
DPWS – Devices Profile for Web Services
HTTP – HyperText Transfer Protocol
HTTPS - HyperText Transfer Protocol Secure
IP – Internet Protocol
IRI – Internationalized Resource Identifier
JME – Java Mobile Edition
OWL – Web Ontology Language
RDF – Resource Description Framework
RDFS - Resource Description Framework Semantics
RIF – Rule Interchange Format
SMTP – Simple Mail Transfer Protocol
SLP – Service Location Protocol
SOA – Service Oriented Architecture
SOAP – Simple Object Access Protocol
TCP – Transmission Control Protocol
UDDI – Universal Discovery Directory and Integration
UDP – User Datagram Protocol
UPnP – Universarl Plug and Play
URI – Uniform Resource Identifier
WSMO – Web Service Modeling Ontology
WSDL – Web Service Description Language
WSML – Web Service Modeling Language
WSMX – Web Service eXecution environment
XML – eXtension Marckup Language

SUMÁRIO

1 INTRODUÇÃO

Redes de computadores se tornaram algo tão comum que podem ser encontradas em toda parte. Exemplos de redes comuns são: a internet, uma rede privada ou até mesmo uma rede telefônica. Essa característica permite que sistemas distribuídos estejam em toda parte. Segundo [Coulouris, 2007], sistemas distribuídos são componentes localizados em computadores ligados a uma rede que se comunicam através de troca de mensagens.

Sistemas distribuídos muitas vezes possuem componentes heterogêneos, e a Arquitetura Orientada a Serviços (SOA) [ERL, 2004] tem se mostrado uma tecnologia eficiente para integrar esse tipo de sistema. SOA permite a criação de componentes de software com baixo acoplamento, que podem ser localizados e invocados dinamicamente por clientes. Essas características são possíveis graças à tecnologia de Web Services [Alonso, Casati, Kuno, and Machiraju 2004] que utiliza alguns padrões que permitem a comunicação entre serviços e a descrição destes, sendo os principais: SOAP [Gudgin et al. 2007], WSDL [Christensen, Curbera, Meredith, Weerawarana, 2001] e UDDI [Riegen, 2002]. SOAP é o formato padrão para troca de mensagens com Web Services, baseado em XML[W3C, 2008]. Uma mensagem SOAP é um documento com formato pré determinado, incluindo envelope e corpo de mensagens. WSDL é um formato também baseado em XML para descrever funcionalidades e serviços. Descrições de serviços em WSDL incluem parâmetros de entrada e saída, protocolos de comunicação, entre outros aspectos dos serviços. UDDI é um repositório para registro e descoberta de serviços.

Utilizando UDDI para encontrar serviços é necessário realizar uma busca no diretório e utilizar os parâmetros de busca corretos para obtenção do serviço. Porém, para que isso ocorra, antes é necessário que o serviço tenha sido registrado. Esse processo não é automatizado, dependendo sempre do fator humano para registrar os serviços e realizar a busca no repositório.

O que se busca é uma automatização do processo de descoberta de serviços, para isso foram criadas diversas tecnologias de descoberta de serviços [Bluetooth, 2009][Jini, 2003][UPnP, 2003][Salutation, 1999]. Essas tecnologias usam diversos mecanismos para realizar essa descoberta. Entre os principais mecanismos utilizados estão consultas *multicast* e serviço de diretórios.

Um fator a ser destacado na descoberta de serviços, é como identificar e classificar os serviços. Uma busca realizada simplesmente por palavras chave é bastante limitada, visto que existe um grande problema em identificar as palavras que descrevem um serviço. Esse tipo de busca simplesmente sintática limita a tecnologia. Uma busca semântica, onde são definidos conceitos comuns e compartilhados, pode melhorar os resultados obtidos e contextualizar melhor o que realmente está sendo buscado.

O processo de descoberta pode ser melhorado através da criação de conceitos que descrevam um serviço. Para descrever esses conceitos utilizamos ontologias[Gruber, 1993]. Assim podemos descrever entidades e relações entre essas entidades de maneira formal, possibilitando que os dados e conceitos sejam interpretados por máquina. Essa técnica nos proporciona uma busca mais ampla e precisa. Serviços descritos dessa maneira são chamados de Serviços Web Semânticos [McIlraith, Son, and Zeng, 2001].

Atualmente é possível acessar e prover serviços através de dispositivos móveis. Dispositivos nos quais se enquadram PDAs, leitores RFID, sensores wireless entre outros. Com capacidade de processamento considerável, conectividade *wireless* e sistemas operacionais complexos, estes dispositivos têm todos os requisitos necessários para fazer parte de uma arquitetura de serviços.

Para que dispositivos móveis possam prover serviços de maneira útil e confiável, temos que levar em consideração alguns aspectos básicos, tais como a intermitência da conexão, limite de recursos como processamento, armazenamento, bateria, entre outros [Forman and Zahorjan, 1994].

Com os problemas listados referentes a serviços móveis, podemos deduzir que é complexo executar Serviços Web Semânticos nesses ambientes. A busca por serviços consome um processamento superior ao que dispositivos móveis podem fornecer, criando mais um desafio para esta área de pesquisa.

Por outro lado, disponibilizar Serviços Web Semânticos móveis contribui para criar um ambiente ubíquo, o qual permitirá um fácil acesso a Serviços Web executando em qualquer ambiente.

Um ambiente ubíquo se trata de um ambiente onde clientes ou usuário podem interagir com dispositivos da maneira mais transparente possível [Weiser, 1993]. Baseado em um perfil de usuário e suas preferências, que podem ser ajustados dinamicamente e automaticamente, serviços são encontrados, disponibilizando funcionalidades que facilitam a vida do usuário.

1.1 MOTIVAÇÃO

As tecnologias presentes atualmente no mercado não atendem de maneira satisfatória os pré-requisitos para criação de um ambiente ubíquo. A implantação de serviços em todos os tipos de dispositivos é um passo importante e essencial para criação desse tipo de ambiente, porém não podemos nos limitar a isso. As descrições sintáticas empregadas nesses serviços atualmente não têm a riqueza da informação necessária para tornar transparente para o usuário as interações com dispositivos no ambiente.

Devemos empregar os conceitos de serviços semânticos para criar um ambiente rico em informações de contexto, assim serviços poderiam identificar necessidades e/ou preferências do usuário sem que ele tenha que realizar uma interação através de uma interface de usuário. Com informações de contexto e sistemas leves e flexíveis, podemos progressivamente diminuir a necessidade de interação do usuário com sistemas computacionais, tornando-as quase imperceptíveis.

Com a integração da semântica em ambientes de serviços móveis, podemos dar um pequeno – mas importante – passo em direção a esse objetivo.

1.2 OBJETIVOS

Nesta sessão são apresentados os objetivos deste trabalho, que são divididos entre objetivo geral e objetivos específicos.

1.1.1 Objetivo Geral

O trabalho descrito nessa dissertação tem como principal objetivo prover um modelo leve e flexível que integre as tecnologias de Serviços Web Semânticos com um sistema de descoberta dinâmica de serviços, focado em ambientes com recursos limitados.

1.1.2 Objetivos Específicos

Os objetivos específicos do trabalho são:

- Inserir dispositivos móveis em um ambiente de serviços semânticos, podendo realizar tanto a descoberta e invocação de serviços, quanto disponibilizar serviços.
- Contribuir para a criação de um ambiente ubíquo, como o previsto por Weiser em 1993.
- Permitir que serviços, depois de descobertos, possam ser invocados sem que seja necessária qualquer intervenção humana, independentemente do ambiente em que estejam localizados.

1.3 JUSTIFICATIVA

A integração das tecnologias utilizadas neste trabalho deve ajudar na criação de um ambiente de computação ubíqua, que pode beneficiar diretamente pessoas em suas casas, carros, escritórios, empresas, etc. Além disso, pode ser de grande ajuda para a indústria em diversos aspectos, como automatização de controle de estoque, monitoramento ativo de ambiente, controle de presença de funcionários, rastreamento de funcionários e equipamentos, etc.

A utilização de padrões e protocolos abertos e amplamente conhecidos tanto no meio acadêmico como corporativo, torna viável a criação de um modelo aberto, leve e flexível para realizar a descoberta semântica de serviços em ambientes com dispositivos móveis.

1.4 METODOLOGIA

Será realizado um estudo da especificação DPWS. Em seguida uma pesquisa na área de Serviços Web Semânticos será realizada, a fim de entender seu funcionamento e identificar a ontologia mais adequada para ser utilizada no modelo.

Um estudo dos trabalhos relacionados será realizado a fim de aprender com esses esforços e identificar oportunidades de contribuições na área de descoberta de serviços.

Tendo conhecimento das tecnologias necessárias e dos trabalhos já realizados, um modelo de descoberta de serviços com descrição semântica leve o suficiente para ser suportado por dispositivos móveis será projetado e implementado. Serão realizados testes com o intuito de realizar medições no tempo de resposta de cada etapa, tanto da busca quanto da invocação de serviços. Será analisado ainda o impacto no tempo de resposta conforme o número de dispositivos conectados aumenta.

1.5 RESULTADOS ESPERADOS

Com a realização deste trabalho esperamos criar um ambiente onde serviços possam ser descritos de maneira semântica e descobertos através da execução de buscas semânticas realizadas em um repositório. Tais serviços devem ser providos e acessados por dispositivos com recursos limitados, mas não limitados a eles.

1.6 ESTRUTURA DO DOCUMENTO

Os demais capítulos dessa dissertação estão organizados como segue. No capítulo 2 é apresentada uma fundamentação teórica dos conceitos e tecnologias utilizados no trabalho. No capítulo 3 são apresentados os trabalhos relacionados, apresentados seus pontos positivos e negativos e por fim comparados. No capítulo 4 é descrita a proposta do trabalho. Informações sobre a implementação e testes de validação do modelo proposto são apresentadas no capítulo 5. Por fim, o capítulo 6 contém as conclusões obtidas com a realização do trabalho e a descrição de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos e tecnologias utilizados no presente trabalho. O capítulo é inicia com os conceitos base de sistemas distribuídos, abordando os aspectos da arquitetura SOA e aplicando-os em dispositivos móveis, chegando até conceitos de Web Semântica.

2.1 SISTEMAS DISTRIBUÍDOS

Sistemas distribuídos são definidos por [Coulouris, 2007] como sendo um conjunto de componentes de hardware ou software interligados por rede e que se comunicam apenas por troca de mensagens. É uma tendência cada vez mais presente, tanto na Web quanto em ambientes empresariais fechados. Sistemas distribuídos possuem características e necessidades específicas. Um fator delicado é a comunicação entre estas aplicações, pois a troca de mensagens deve ocorrer de maneira eficaz e confiável, de modo a garantir o bom funcionamento de tais sistemas. Esta é uma tarefa bastante complexa, levando em consideração os requisitos necessários para que isso ocorra.

A implementação de sistemas distribuídos nos levam a lidar com algumas questões que caso não sejam tratadas podem gerar problemas, questões como: incompatibilidade de protocolos de comunicação, linguagem de programação, falta de interfaces bem definidas, tratamento de falhas, concorrência. Daí surge a necessidade da criação de padrões de comunicação para que estes sistemas possam interoperar e atingir seus objetivos de maneira satisfatória.

Um dos principais motivos para se utilizar sistemas distribuídos é o compartilhamento de recursos, tanto de hardware quanto de software.

2.1.1 Computação Móvel

A computação móvel é possível graças a dispositivos portáteis conectados em redes sem fio, como WiFi, GPRS, 3G, Bluetooth, infravermelho ou até mesmo radiofrequência [Coulouris, 2007]. Essa conec-

tividade permite que pessoas continuem conectadas, mesmo quando em movimento [Forman and Zahorjan, 1994]. Computação móvel provê uma forma eficiente e flexível de comunicação para seus usuários.

Os dispositivos móveis facilitam nossas vidas. Porém, existem alguns cuidados especiais que devemos ter ao desenvolver aplicações para estes dispositivos. Normalmente, por serem portáveis, estes possuem diversas limitações, como: memória, processamento e energia. Existem diversas ferramentas para desenvolver sistemas para dispositivos móveis que são otimizadas para lidar com essas limitações. Uma das tecnologias mais utilizadas para este fim é o JME [Oracle, 2011].

2.1.2 Computação Ubíqua e Pervasiva

Computação Ubíqua foi citada pela primeira vez por Mark Weiser em 1993. Naquela época, Weiser e seus colegas da Xerox já tinham uma visão de como funcionaria um ambiente com esse paradigma, porém ainda existiam diversas barreiras tecnológicas.

Pode-se dizer que computação ubíqua se trata de um ambiente saturado de recursos computacionais e de comunicação integrados com usuários humanos [Satyanarayanan, 2001]. Para que a computação ubíqua se torne uma realidade temos alguns problemas a serem resolvidos, sendo os quatro principais:

- o uso efetivo dos espaços inteligentes – espaços bem definidos onde os dispositivos atuam;
- invisibilidade dos dispositivos, para que o ambiente computacional fique o mais transparente possível ao usuário;
- escalabilidade localizada, ou seja, o ambiente do usuário fica cada vez mais rico em dispositivos e o número de interações aumenta gradativamente, elevando principalmente o consumo de recursos de rede e energia;
- mascaramento da irregularidade condicionada, pois o nível de penetração de dispositivos pervasivos será diferente em diferentes ambientes, portanto é necessário encontrar uma maneira para equilibrar isso, pois uma distribuição uniforme pode levar anos.

2.1.3 SOA

Service Oriented Architeture (SOA) é um arquitetura de sistemas distribuídos, na qual os sistemas ficam distribuídos como serviços e são acessados através de um protocolo padrão. Essas características da arquitetura são possíveis graças à tecnologia de *Web Services* [Singh et al., 2004]. Devemos ser capazes de encontrar os serviços disponíveis com certa facilidade, e se possível de forma automatizada. Para isso, provedores de serviço disponibilizam a interface de seus serviços em um repositório. Dessa forma, clientes podem realizar buscas por serviços no repositório, de modo que posteriormente, de posse das informações necessárias para acessar o serviço, o cliente possa facilmente invocá-lo. Ou seja, a arquitetura SOA permite que componentes de software sejam publicados, descobertos e invocados por um usuário ou mesmo outro componente [Ayala et al. 2002].

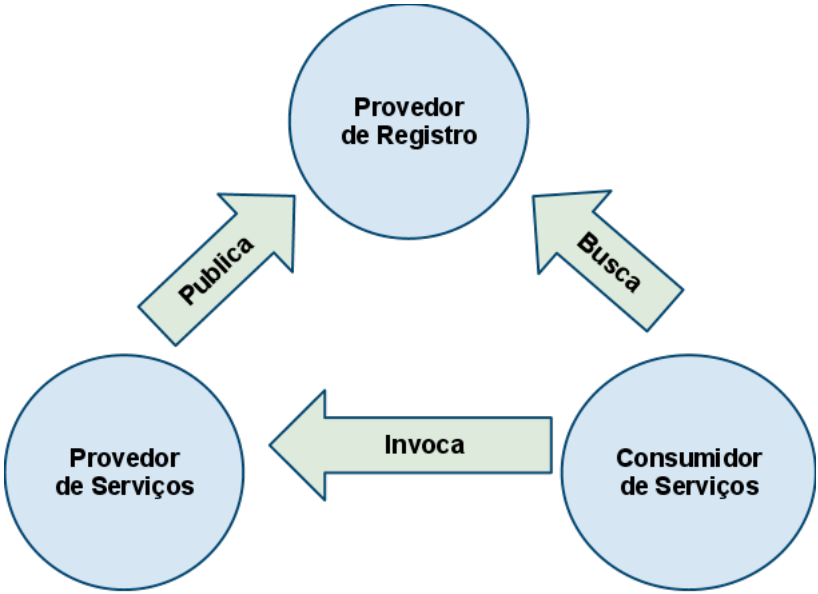

Figura 1 – Arquitetura SOA

Esta arquitetura possibilita o desenvolvimento de sistemas com baixo acoplamento e alta coesão. O cliente não precisa conhecer os detalhes de funcionamento interno do serviço. Em algumas aplicações o sigi-

lo sobre o funcionamento interno da aplicação não se trata apenas de uma opção, é um requisito, como por exemplo, em sistemas bancários, que disponibilizam serviços para o cliente via Web.

Na figura 1 podemos visualizar como funciona a arquitetura.

O provedor de serviços é quem disponibiliza serviços para que o consumidor possa utilizá-los. Ele publica seus serviços no provedor de registros. O provedor de registros é responsável por manter catalogados todos os serviços publicados. O consumidor de serviços, que é quem utilizar algum serviço para realizar uma ação, busca um serviço no provedor de registro e posteriormente invoca o serviço diretamente no provedor de serviços.

2.1.4 Web Services

Web Services foram concebidos com intuito de promover um padrão que permitisse além da comunicação entre aplicações distribuídas, também a interoperabilidade entre elas. Pode-se definir *Web Service* como um sistema computacional acessível na *Web* (ou redes privadas) através de uma URI, acessada através de protocolos baseados em XML. Essa tecnologia foi desenvolvida para prover suporte à interoperabilidade em interações entre sistemas computacionais distribuídos [SINGH et al. 2004]. *Web Services* independem de linguagem de programação ou *framework* de desenvolvimento, possibilitando a comunicação entre aplicações heterogêneas. Isso é possível graças aos padrões de comunicação adotados por esta tecnologia.

Padrões amplamente difundidos na Web, como XML, HTTP, SMTP são utilizados. Cada *Web Service* possui uma descrição, na qual pode-se encontrar os serviços que estes sistemas possuem, assim como os parâmetros que os serviços precisam para realizar determinada tarefa e, eventualmente, retornar um resultado a quem o invoca. Essa descrição utiliza o padrão WSDL, o qual traz além da descrição das funções do *Web Service*, também informações sobre os protocolos de comunicação utilizados e as informações necessárias para a invocação de seus serviços. O WSDL é um formato para documentos XML com as informações acima descritas, mas não limitado a elas.

Após obter a descrição de um *Web Service*, um cliente pode invocar seus serviços. Essa invocação ocorre utilizando o padrão SOAP, e na maioria das vezes o protocolo de comunicação utilizado é o HTTP. As informações a serem enviadas pelo cliente são encapsuladas por um en-

velope SOAP, o qual consiste em um arquivo XML que, além das informações destinadas ao *Web Service*, pode conter outras informações em seus cabeçalhos (segurança, roteamento, etc). Então o pacote é enviado ao *Web Service* através de um protocolo de comunicação suportado pelo serviço. O *Web Service* recebe a requisição, realiza o processamento necessário e pode retornar uma resposta ao cliente da mesma maneira.

Para que não seja necessário conhecer previamente o endereço de rede de cada *Web Service*, existe mais um padrão, chamado UDDI, que consiste em um repositório onde os *Web Services* são registrados por seus criadores. O repositório possui as informações dos *Web Services* registrados, e entre essas informações encontra-se o WSDL. Dessa maneira os clientes consultam o repositório UDDI para buscar por serviços que eventualmente necessitem, obtêm a descrição do *Web Service* que tem o serviço disponível e pode acessá-lo sem ter conhecimento prévio sobre a existência nem localização do serviço.

2.1.4.1 WSDL

A W3C especifica o WSDL como sendo um padrão que descreve serviços disponibilizados na rede. Estes serviços são considerados pela W3C interfaces programáticas disponibilizadas na rede para executar determinada função. WSDL descreve esses serviços como sendo um conjunto de *endpoints* que se comunicam através da troca de mensagens. Uma descrição WSDL é um documento XML que define alguns conceitos abstratos para descrição de características de um serviço. Algumas dessas características são abstratas como: *message, operation* e *port type*. Essas características representam respectivamente, uma representação dos dados a serem trocados, a descrição de uma operação suportada pelo serviço e o conjunto de operações suportado por um ou mais *endpoints*.

Além dessas características abstratas temos ainda algumas outras concretas: *types,* que contém as definições dos tipos de dados usados; *binding* contém o protocolo e a especificação do formato de dados de um *port type*; *port* possui informação de um *endpoint,* combinando o endereço de rede e um *binding;* e *service é* uma coleção de *endpoints* relacionados.

2.1.4.2 SOAP

Em 2002 a W3C definiu a versão 1.2 do protocolo SOAP. Nesse documento SOAP foi definido como sendo um protocolo leve com a intenção de troca de informação de maneira estruturada em ambientes descentralizados e distribuídos. SOAP usa XML para criação de mensagens e protocolos da camada de aplicação, principalmente HTTP. SOAP provê um *framework* de mensagens básico que pode formar a camada base de um *Web Service*.

Uma mensagem SOAP é composta basicamente por três componentes: o envelope que envolve os outros componentes, o cabeçalho que possui informações de como a mensagem deve ser processada, e o corpo que possui as informações da mensagem.

2.1.4.3 UDDI

UDDI se trata de um repositório onde *Web Services* são registrados para poderem ser descobertos posteriormente. É um padrão recomendado pela OASIS para registro e descoberta de serviços. Em 2004 foi especificada a versão 3.0.2 com o objetivo de facilitar o registro e acesso de *Web Services* criados.

2.1.5 DPWS

O DPWS [Microsoft 2006] é um padrão proposto pela Microsoft em 2006 e posteriormente adotado pelo OASIS. Este padrão define uma infra-estrutura de comunicação que permite dispositivos conectados a uma rede ter um comportamento similar a dispositivos *plug-and-play*. Dispositivos com estas características podem ter seus serviços descobertos e invocados de maneira automatizada, ou seja, sem a interação humana.

O padrão DPWS é baseado na tecnologia de *Web Services*, utilizando-a para a comunicação entre dispositivos. Além dos mecanismos usuais presentes em *Web Services*, é utilizado também um mecanismo para localização dinâmica de serviços. Aplicações que implementam esse padrão não são limitadas por linguagens de programação ou sistema operacional. Qualquer dispositivo com conectividade e recursos de processamento -

como computadores móveis, *tablets*, sensores e muitos outros - podem implementar este padrão e assim acessar e prover serviços.

A pilha de protocolos empregada para o funcionamento do DPWS pode ser visualizada na figura 2.

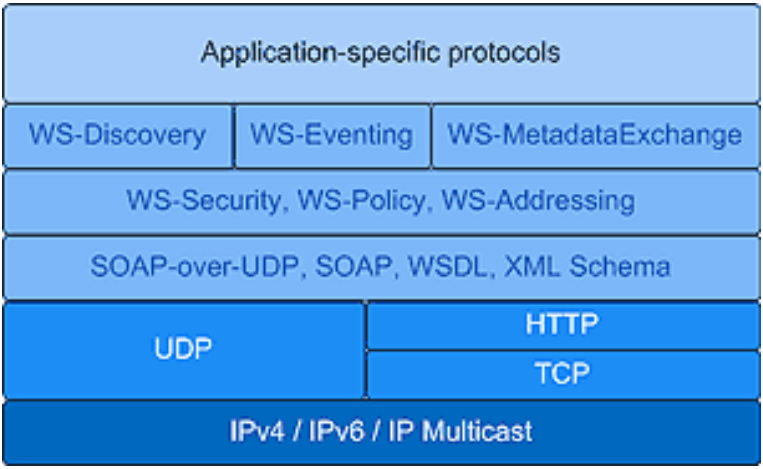

Figura 2 - Pilha de protocolos DPWS
Fonte: http://ws4d.e-technik.uni-rostock.de/dpws/

DPWS utiliza em sua pilha de protocolos diversos protocolos já estabelecidos. Nas camadas inferiores temos os protocolos padrão de rede, IP, TCP, UDP e HTTP, na camada imediatamente superior estão os padrões de *Web Services* recomendados pela W3C e acima algumas outras especificações de *Web Services*, cada uma com uma função específica para viabilizar a arquitetura de maneira padronizada.

Os serviços do DPWS são divididos em dois tipos: *hosting services*, que representam o dispositivo no qual serviços são hospedados, e provêem informações sobre o dispositivo, permitindo assim a sua descoberta. São basicamente um contêiner para outros serviços; e *hosted services*, que são os serviços providos por um dispositivo, ou seja, são quem realmente executa as operações. *Hosted services* são hospedados pelos *hosting services*. Serviços têm seu próprio endereço de rede para que possam ser acessados por outros dispositivos.

O fluxo de mensagens do DPWS entre um cliente e um provedor de serviço pode ser visualizado na figura 3. Um cliente envia via *broadcast* uma requisição *Probe* com o objetivo de encontrar outros dispositivos com características especificadas no corpo da mensagem. Um dispositivo que é compatível com a busca responde com uma mensagem do tipo *Probe Match*. Depois de identificar um ou mais dispositivos, o cliente deve enviar uma mensagem do tipo *Get Metadata* para receber mais informações dos dispositivos

encontrados, incluindo a lista de serviços disponíveis. De posse dessa informação, o cliente envia novamente uma mensagem do tipo *Get Metadata,* mas dessa vez diretamente para os serviços hospedados a fim de obter as informações necessárias para a invocação destes serviços.

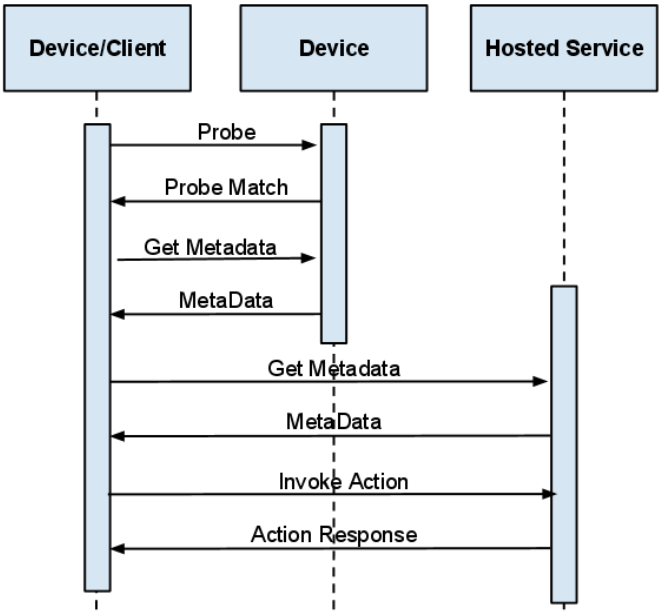

Figura 3 - Fluxo de uma requisição DPWS

2.2 WEB SEMÂNTICA

Segundo Berners-Lee, Hendler e Lassila [2001], Web Semântica é considerada a próxima geração da Web. A Web Semântica adiciona significado aos dados da Web através de anotações, de maneira que agentes inteligentes sejam aptos a entender esse significado. Para esta realização, ontologias são amplamente utilizadas.

Atualmente a W3C é quem define as especificações da Web Semântica. Na figura 4 podemos visualizar as camadas recomendadas pela organização para a implementação da Web Semântica.

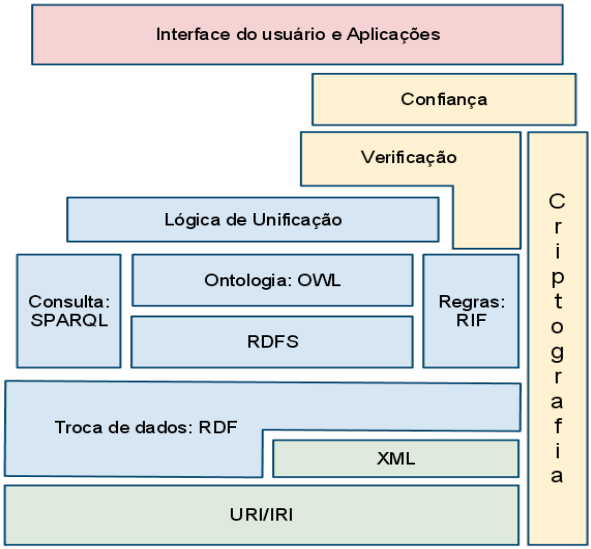

Figura 4 - Camadas da Web Semântica

As camadas URI/IRI e XML já fazem parte dos padrões Web e fornecem uma base sintática para as linguagens da Web Semântica. Na camada de troca de dados temos o padrão RDF que é descrito na sessão 2.2.2.1, logo acima temos o RDFS que faz parte da especificação do RDF e provê uma estrutura para definir linguagens. Ainda temos a camada de regras, que usa o padrão RIF [Pollers, Boley, Kifer 2010], que se trata de um formato para regras de troca de dados. Na camada de ontologias o padrão recomendado é OWL, que é descrito na sessão 2.2.2.2. A realização de consultas é feita através de SPARQL [Prud'hommeaux and Seaborne, 2008], uma linguagem criada para a realização de consultas em RDF. Sobre essas camadas existe uma camada para unificação da lógica contida nas camadas inferiores. A camada *criptografia* provê uma infra-estrutura de criptografia para a camada *confiança*, que junto com a camada *verificação,* que não são bem definidas na especificação, mas devem ser utilizadas para garantir a confiabilidade dos dados apresentados. A camada *confiança* utilizada junto com criptografia e assinatura digital, e com a prova formal fornecida pela camada *verificação* devem garantir a integridade e a origem dos dados.

2.2.1 Ontologias

Ontologias são usadas na filosofia como uma teoria sobre a natureza da existência, "quais os tipos de coisas existem" [Gruber, 1993]. Na computação uma ontologia é quem define formalmente os elementos existentes e o relacionamento entre eles em um sistema. Normalmente uma ontologia para Web possui uma taxonomia e um conjunto de regras de inferência [Fensel et al, 2007].

A taxonomia define classes de objetos e a relação entre elas. Relacionamentos como generalização, especialização são essenciais para expressar as entidades Web. Regras de inferência provêem um poder de expressividade adicional a essa estrutura. Com essas regras podemos fazer relações de composição e agregação [Berners-Lee, Hendler, and Lassila, 2001].

Normalmente ontologias computacionais são representadas através de um arquivo XML que pode ser facilmente lido e interpretado por computadores. O uso de ontologias promove um entendimento comum e compartilhado sobre um domínio, e nos permite lidar com problemas semânticos [Falbo, 2004].

O mapeamento dos conceitos que conhecemos para ontologias normalmente é feito através de domínios específicos, por exemplo, uma ontologia para mapear a anatomia humana, uma ontologia para mapear componentes eletrônicos, uma para mapear datas e medidas de tempo. Tais ontologias podem ter conceitos comuns, importar conceitos umas das outras ou mesmo ser estendidas.

2.2.2 **Aplicando ontologias na Web Semântica**

[Fensel, 2003] considera que a característica principal das ontologias é representar terminologias consensuais e semânticas do mundo real formalmente, de maneira que o entendimento humano e de máquinas seja comum. Essa característica permite que conceitos e relações sejam compartilhados e reusados.

Ontologias diferentes podem mapear um mesmo conceito de maneira diferente. Aos olhos de uma pessoa seria muitas vezes facilmente identificável que duas descrições se tratam de um mesmo conceito. Contudo, na área da computação isso pode gerar algum tipo de confusão. Existem técnicas para que conceitos iguais, descritos de maneira dife-

rente, possam ser automaticamente identificados, porém não estão presentes em todos os modelos ontológicos.

2.2.2.1 RDF

RDF é a primeira linguagem criada para a Web Semântica [Klyne and Carroll, 2004]. Foi desenvolvido para adicionar metadados compreensíveis por máquina ao conteúdo já existente na Web.

A especificação do RDF recomenda que se use XML para serialização, possibilitando a troca de informações entre aplicações. Documentos RDF são estruturados em triplas que consistem em sujeito, predicado e objeto. Um sujeito se relaciona com um objeto através do predicado. Essas triplas representam as relações entre os conceitos descritos pela linguagem.

RDF apresenta algumas primitivas ontológicas para representação de conceitos como: classes, propriedades e instâncias. Para relacionar esses conceitos, existem algumas primitivas de relacionamento: *instance-of*, *subclass-of* e *subproperties-of*.

2.2.2.2 OWL

OWL (*Web Ontology Language)*, atualmente em sua versão 2, é uma linguagem para a Web Semântica com um significado formalmente definido [Bechhofer, 2004]. A especificação da ontologia apresenta conceitos de classes, propriedades, indivíduos e valores de dados.

OWL é uma linguagem de ontologia que estende o RDF, aumentando sua expressividade. OWL é subdividida em três sublinguagens conforme sua expressividade: *Lite*, DL e *Full*.

Na referência de sublinguagens de OWL [Bechhofer, 2004], temos as seguintes descrições de cada uma das três acima citadas:

- OWL *Lite* é uma linguagem que suporta um subconjunto das construções de OWL DL, foi criada para ser uma linguagem simples e prover um subconjunto funcional de instruções para iniciar o uso de OWL.

- OWL DL (*Description Logic*) é uma linguagem mais completa com restrições apenas no uso de características do RDF. Quando RDF é usado, OWL DL requer que sejam separadas classes, propriedades, indivíduos e valores de dados.
- OWL *Full* possui todas as características de OWL DL, porém não impõe restrições no uso de RDF.

2.2.2.3 OWL-S

OWL permite representar diversos aspectos da Web de maneira semântica, possibilitando identificar contextos em que o usuário, sendo humano ou um sistema, esteja inserido. Contudo, segundo Martin et al. [2004], a Web Semântica deve prover além de dados, serviços. O desejo de localizar, selecionar, invocar entre outras operações automaticamente, motivou a criação de OWL-S.

OWL-S é descrito em [Martin et al. 2004] como sendo uma linguagem para descrever serviços, provendo um vocabulário que possa ser utilizado em conjunto com OWL.

Em 2008 foi finalizada a versão 1.2 da ontologia [Martin et al., 2005], criada por um grupo de pesquisadores chamado OWL-S *Coalition*.

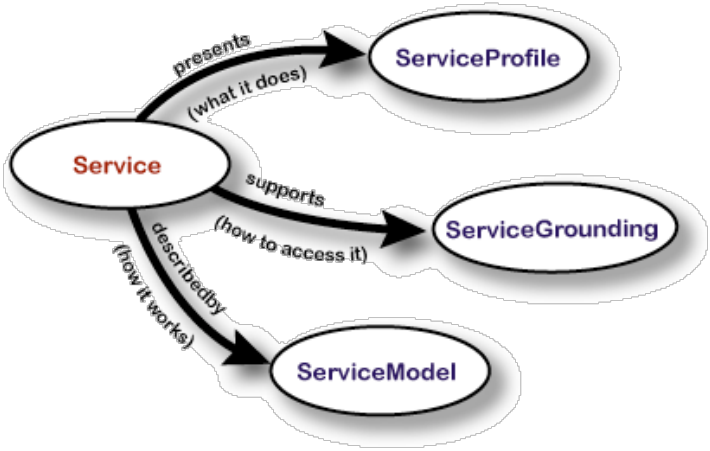

Figura 5 - Estrutura de um serviço descrito em OWL-S
Fonte: http://www.ai.sri.com/daml/services/owl-s/1.2/overview/ServiceOntology1.1.gif

OWL-S descreve serviços levando em consideração 3 aspectos: *presents* que descreve o que o serviço faz do ponto de vista do usuário, *supports* descreve como acessar o serviço e *describedby* que descreve como o serviço funciona. Na figura 5, podemos visualizar uma representação da descrição de um serviço.

As classe *ServiceProfile* provê as informações necessárias para a descoberta do serviço, enquanto *ServiceModel* e *ServiceGrounding* juntas fornecem a informação necessária para o cliente realizar o uso do serviço.

2.2.2.4 WSMO + WSMF

Web Service Modeling Ontology é a base conceitual de WSMF (*Web Service Modeling Framework*) [Fensel and Bussler, 2002]. WSMO tem como objetivo descrever todos os aspectos relevantes de *Web Services* com o intuito de automatizar o máximo das tarefas que envolvem esses serviços, como descoberta, automação, composição, monitoramento, etc. [Roman et al. 2005].

WSMO define quatro principais elementos a fim de descrever Serviços Web Semânticos: *ontologies, Web Services, goals e mediators* [Fensel at. al., 2007].

- *Ontologies* provêem a terminologia usada pelos outros elementos para descrever aspectos relevantes do domínio. Diferentemente de terminologias que focam apenas em descrições sintáticas, esse conceito busca uma descrição que dê algum significado a aplicações e componentes. Além disso, essas definições formais devem ser compreensíveis por máquina.
- *Web Services* representam entidades computacionais que possam prover algum serviço, o qual agrega algum valor a um domínio. Aspectos desse serviço, como interfaces, o que ele faz e como funciona internamente, são descritos pela terminologia provida pela ontologia.
- *Goals* descrevem aspectos relacionados a desejos dos usuários, no sentido de identificar qual serviço eles desejam. Assim como *Web Services*, são descritos pela terminologia provida pelas ontologias.
- *Mediators* têm como objetivo lidar com as diferenças de terminologias utilizadas, promovendo a interoperabilidade entre diferentes

elementos do WSMO. As diferenças tratadas podem estar na ontologia em nível de dados, ou nos *Web Services* nos níveis de processo ou protocolo.

A linguagem usada para criar os documentos que descrevem os elementos do modelo ontológico se chama *Web Service Modeling Language* (WSML) [DERI, 2008]. WSML é uma linguagem baseada em XML. Os recursos são descritos de maneira isolada, evitando acoplamento e mantendo-os distribuídos.

Documentos WSML que descrevem recursos são executados em um ambiente que implementa o WSMO, os recursos são enviados e armazenados para que possam ser utilizados posteriormente. *Web Service Modeling Execution* (WSMX) [DERI, 2008] possui uma arquitetura que permite integrar sistemas existentes, através da criação de componentes que façam o intermédio necessário para integração, evitando assim alterações no software existente e mantendo o ambiente o máximo possível desacoplado. Na figura 6 é apresentada a arquitetura do WSMX.

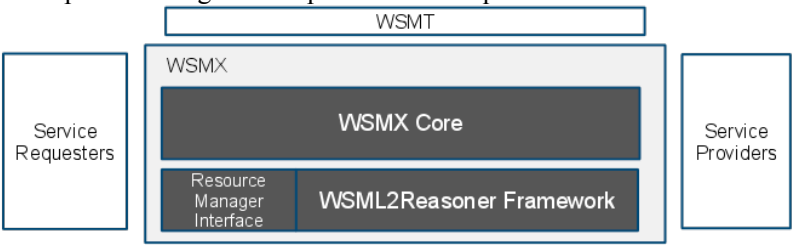

Figura 6. Arquitetura do WSMX

Na parte superior da arquitetura do WSMX temos o *Web Service Modeling Toolkit* (WSMT) [DERI, 2005], que é uma ferramenta baseada na plataforma Eclipse para acessar as funcionalidades do ambiente de execução e criar elementos WSMO através de uma interface gráfica. Dentro do WSMX temos alguns componentes: o *Core*, que possui a base do sistema, e onde *plugins* podem ser acoplados; a interface de gerenciamento de recursos, que é responsável por gerenciar cada novo recurso adicionado; e por fim o *WSML2Reasoner Framework* [Grimm, Keller, Lausen and Nagypal, 2007], que é um *framework* de processamento semântico, responsável por processar os *goals* e encontrar os dados dos serviços desejados.

2.3 SERVIÇOS WEB SEMÂNTICOS

Serviços Web Semânticos consistem da aplicação de *Web Services* ao conceito de Web Semântica. Serviços descritos de maneira complexa a fim de dar significado aos dados, possibilitando um menor acoplamento, mediação de diversos aspectos dos serviços e permitir que essas descrições sejam compreensíveis por máquina [Fensel et al., 2007].

Aplicando a semântica a Serviços Web, pretende-se tornar viável a operação automática ou semi-automática desses serviços. Descoberta, invocação, monitoramento, implantação, composição entre outras operações estão incluídas nesse objetivo [Mcllraith, Son and Zeng, 2001].

Serviços Web Semânticos ainda são uma tecnologia em desenvolvimento. Apesar dos recursos desejáveis, a falta de definição de padrões e ferramentas para criação dos serviços dificulta sua utilização.

A automatização do processo de descoberta de serviços, permitindo que sistemas computacionais encontrem os serviços que necessitam e efetuem suas tarefas sem a intervenção humana e a composição de serviços para efetuar uma tarefa complexa, seriam extremamente favorecidas com a utilização dessa tecnologia. Ambas as características beneficiariam diretamente nosso dia a dia. Atualmente um cenário comumente citado na literatura em que a composição de serviços se aplica é no planejamento de viagens. Um cenário bastante restrito e específico nos permite realizar esse tipo de operação com as tecnologias atuais, porém com Serviços Web Semânticos poderíamos agregar essa facilidade a diversos outros serviços.

Segundo Fensel et al. [2007] os setores mais beneficiados com a tecnologias de Serviços Web Semânticos seriam o comércio eletrônico e as grandes empresas que necessitam da integração de suas aplicações. Tornando as descrições de serviços compreensíveis por máquina, a intervenção humana deixaria de ser necessária, e o processo de integração aconteceria em uma velocidade bastante superior.

A maneira utilizada atualmente para tornar os dados compreensíveis por máquinas é a criação de conceitos utilizando ontologias. Esses conceitos são utilizados para descrever aspectos de serviços de maneira compartilhada. As principais linhas de pesquisa consideram OWL-S e WSMO como ontologias para a criação desses conceitos. Ambas são bastante expressivas e possuem a capacidade para suprir tais necessidades.

2.4 CONSIDERAÇÕES

No presente capítulo são apresentados conceitos e tecnologias, que são essenciais a para o desenvolvimento da presente dissertação. O entendimento básico da evolução de sistemas distribuídos, chegando até a computação móvel e protocolos de descoberta de serviços. Tanto quanto web semântica e descrição formal e compartilhada de componentes e recursos computacionais, de maneira que estes sejam compreensíveis por máquina.

O objetivo deste capítulo é prover uma base de conhecimento, possibilitando o entendimento por parte do leitor do funcionamento e da contribuição dos modelos apresentados posteriormente neste trabalho de mestrado. São enfatizados a descoberta de serviços e a descrição semântica de serviços utilizando WSMO, os quais são o principal foco da dissertação.

Web Services Semânticos ainda são uma tecnologia em desenvolvimento. Não temos padrões definidos e os modelos ontológicos criados para a descrição dos serviços, apesar de ter toda uma base conceitual, possuem uma implementação que deixa muito a desejar. As ferramentas e os ambientes de execução estão em testes e possuem pouca ou nenhuma documentação, dificultando o desenvolvimento de novos trabalhos nessa área.

3 TRABALHOS RELACIONADOS

Existem alguns protocolos já estabelecidos para descoberta de serviço, como Jini, OSGi [OSGi Alliance, 2010], SLP [The Internet Engineering Task Force, 1999], UPnP, Salutation, DPWS etc. Contudo, estes possuem limitações como: linguagem de programação, descrições unicamente sintáticas, funcionam apenas com redes locais, não podem ser executados por dispositivos móveis, etc. Existem diversos esforços no sentido de estender a capacidade destes protocolos para aprimorar o processo de descoberta, ou mesmo ampliar a abrangência destes protocolos. Neste capítulo são apresentados brevemente os principais trabalhos semelhantes a este, citando seus pontos fracos, fortes e fazendo uma comparação entre eles. Por fim temos uma visão geral do que o presente trabalho acrescenta em relação aos outros trabalhos.

3.1 DSB

Device Service Bus [Medeiros e Siqueira, 2009] é um modelo de descoberta de serviço baseado no DPWS. Possui uma arquitetura leve com a intenção de ser executado em ambientes com recursos limitados. Tem como principal objetivo integrar diferentes tecnologias de comunicação. Através de *plugins*, o DSB é capaz de prover e acessar serviços utilizando protocolos de comunicação heterogêneos.

A arquitetura do DSB é formada por diferentes componentes, os quais possuem funções específicas. O DPWS é suportado através do *device tunnel*, que é um serviço DPWS, através do qual o sistema pode encontrar outros *device tunnels* ou dispositivos compatíveis com DPWS. Serviços não compatíveis com DPWS são suportados através do *virtual device*, que é um componente que provê uma interface DPWS para dispositivos que não suportam tal tecnologia. *Virtual devices* se comunicam com dispositivos DPWS através do componente chamado *Bridge*, que para executar tal tarefa mantém um *cache* do *Virtual Device*. Tecnologias de comunicação diferentes são suportadas através de *Converters*, que conhecem detalhes da implementação de dispositivos e seus serviços. Cada *Converter* é responsável por uma tecnologia de comunicação, fazendo a tradução de requisições de diferentes protocolos de comunicação a fim de que todos os componentes se comuniquem.

Inicialmente focado em Wi-Fi, Bluetooth e RFID, o DSB provê interfaces para expansão e integração com outras tecnologias. Contudo o trabalho é focado na integração dessas diversas tecnologias e não apresenta esforços no sentido de integração semântica, sendo um sistema com serviços descritos apenas sintaticamente. Isso resulta em uma grande necessidade de interação homem-máquina a fim de identificar e invocar o serviço procurado.

3.2 HOME SOA

Home SOA [Bottaro e Gérodolle, 2008] foi projetado para ser utilizado em redes domésticas, mais especificamente em automação residencial. Através do que denomina *drivers* refinados, o *Home SOA* trata as diferenças entre protocolos, permitindo que dispositivos que utilizam diferentes protocolos de comunicação interajam.

Serviços são criados utilizando a tecnologia OSGi, que permite a criação de *bundles*, que podem prover e consumir serviços. Esses *bundles* importam uma *API* de serviços que permite que estes realizem as operações de registro consulta e invocação de serviços. O provedores de serviços registram seus serviços em um repositório onde posteriormente estes podem ser descobertos.

É utilizada semântica para descrição dos dispositivos e os serviços oferecidos, contudo o *middleware* em si não apresenta um modelo semântico próprio, mas sugere que um modelo semântico criado por terceiros seja agregado ao sistema. Portanto, a forma como estes serviços são descritos vai depender do tipo de integração realizado.

A comunicação do modelo é realizada através da tecnologia OSGi, pois no trabalho julga-se que *Web Services* não são ideais para tal ambiente, visto que a comunicação gera um grande tráfego. Contudo, fazendo essa escolha o modelo fica limitado a linguagens suportadas pela máquina virtual Java.

3.3 AIDAS

O projeto AIDAS [Toninelli, Corradi, Montanari, 2008] é um *middleware* que utiliza dados semânticos para descrever serviços. É dividido em dois conjuntos lógicos: o gerenciamento de descoberta, que

provê as funcionalidades necessárias para descoberta e seleção de serviços baseado em informações de contexto de usuário e preferências do usuário; e o gerenciamento de configuração, que provê uma infraestrutura para que dispositivos portáteis possam divulgar suas funcionalidades semânticas para outros dispositivos.

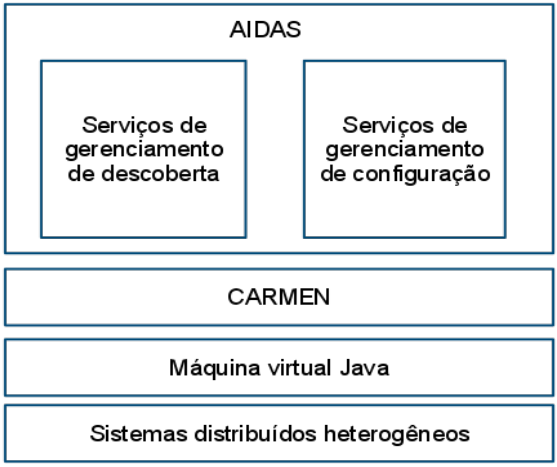

Figura 7 – Arquitetura do AIDAS

AIDAS é desenvolvido sobre o sistema CARMEN [Bellavista, Corradi, Montanari e Stefanelli, 2003], sistema baseado em Java que cria *shadow proxies* que, caso o dispositivo que vá prover um serviço tenha limitações de recurso, agem automaticamente em seu lugar. O CARMEN provê *proxies* com ambientes de execução que são chamados de *places*. Os *places* normalmente modelam nodos que podem ser agrupados em domínios.

Na Figura 7 pode ser visualizada de maneira superficial a arquitetura do *middleware* AIDAS. Como base, temos um sistema operacional qualquer, logo acima é executada a máquina virtual Java, e finalmente o sistema CARMEN, sobre o qual o AIDAS é executado. Dentro do AIDAS pode-se identificar seus dois principais componentes, o sistema de gerenciamento de descoberta e o sistema de gerenciamento de configuração.

Para o objetivo de criar um ambiente ubíquo, o AIDAS possui algumas limitações. A primeira é o suporte limitado a dispositivos móveis, os quais não provêem serviços diretamente. Portanto, a descoberta e

invocação de serviços a partir de dispositivos móveis não é uma realidade para o projeto.

Outra limitação do modelo é que, se limita à tecnologia Java, não suportando serviços em ambientes heterogêneos, o que seria essencial para a criação de um ambiente ubíquo.

3.4 P2P-BASED SEMANTIC SERVICE MANAGEMENT IN MOBILE AD-HOC NETWORKS

Este trabalho [Baumung, Penz e Klein, 2009] apresenta um modelo semântico de gerenciamento de serviços, que se trata de uma rede *ad-hoc* se comunicando através de protocolos P2P.

O sistema é constituído por uma base, que provê a infraestrutura necessária para comunicação, englobando infraestrutura de rede e protocolos de comunicação. Logo acima da base temos o conteúdo do *middleware*, que é subdividido em três camadas: *peer to peer - multicast*, *pool – manager* e processamento semântico.

- *Peer to Peer – multicast*: é a camada inferior da pilha, e provê uma estrutura eficiente para a realização de requisições de serviço através de *multicast*. Os serviços são agrupados em redes móveis *ad-hoc*.
- *Pool – manager*: nessa camada é tratado o gerenciamento de serviços. As requisições são recebidas e os provedores de serviços são propriamente encontrados. Para minimizar a número de requisições necessárias para manter a rede atualizada, foi criado um mecanismo de *cache* que mantém informações atualizadas dos prestadores de serviço.
- Processamento semântico: a camada superior tem como objetivo viabilizar de maneira transparente o uso de serviços, através da descoberta, seleção e invocação de serviços. Os serviços são descritos através de um conjunto de possibilidades de mudanças de estado, e as descrições semânticas são feitas utilizando a *DIANE Service Description Language (DSD)* [Klein, Ko¨nig-Ries e Mu¨ssig, 2005].

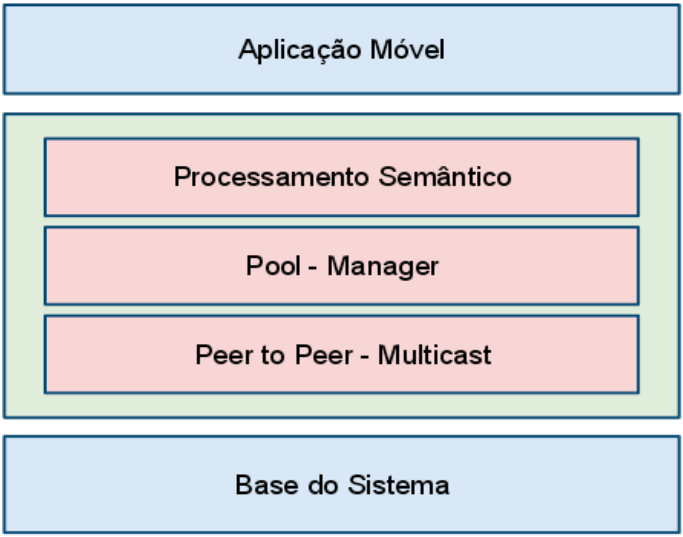

Figura 8 – Camadas do P2P

Acima destas camadas é implementada uma aplicação móvel, onde está contida uma interface gráfica para que o usuário possa interagir com o *middleware* e localizar os serviços desejados. Todas essas camadas podem ser melhor visualizadas na Figura 8.

Limitado a redes *ad-hoc*, possui uma capacidade não muito grande de dispositivos. Todas as requisições são centralizadas em um *pool*, portanto a fim de evitar uma inundação na rede todas as requisições são centralizadas. O *pool* é construído baseado no protocolo SLP.

3.5 CONSIDERAÇÕES

Cada trabalho cobre uma área específica, porém não são flexíveis o suficiente ou não levam em consideração algum aspecto importante para a criação de um ambiente ubíquo. O modelo proposto no trabalho tenta cobrir todos estes aspectos, que podem ser visualizados na tabela 1.

Tabela 1- Comparação dos trabalhos relacionados

Modelo	Comunicação	Restriços de linguagem/ambiente	Escopo	Suporte a Semântica	Suporte a Mobilidade
AIDAS	WS	Java	qualquer um	Parcial	Limitado
HOME SOA	OSGi	Java	mídia residencial	Componentes externos	Total
DSB	WS	nenhuma	qualquer um	Nenhum	Total
P2P-Based Semantic Service	SLP	Suporte a SLP	Redes Ad Hoc	Total	Requisições centralizadas

Apesar do AIDAS utilizar Serviços web para comunicação, seus serviços executam sobre uma pilha que possui componentes diretamente dependentes da máquina virtual java, o que ocasiona limitações em seu ambiente de execução. O suporte a semântica tem uma limitação no momento da busca, onde usa-se informações do contexto do usuário. Porém, não o trabalho não provê uma interface para que o usuário especifique parâmetros referentes ao serviço. A mobilidade é limitada devido a criação de *proxies* onde os serviços são emulados, esta técnica permite uma maior disponibilidade do serviço e ajuda a resolver o problema da intermitência da conexão, contudo dispositivos como sensores ficam com seus serviços prejudicados nesse modelo.

Home SOA se comunica através de OSGi, protocolo ligado diretamente a máquina virtual java, limitando os serviços criados a esta, apesar de sugerir a suporte a semântica o modelo não apresenta nenhuma solução que suporte esta tecnologia, apenas indica que é possível acoplar um componente que proveja suporte a semântica, fazendo com que a descrição e descoberta sigam o padrões do componente acoplado.

DSB é um modelo bastante leve e flexível, baseado em DPWS, almeja a integração entre diferentes tecnologias de comunicação. O suporte a uma descrição semântica não é abordado no modelo.

P2P-Based Semantic Service utiliza o protocolo SLP. A única restrição na questão de ambiente de execução/linguagem de programação é que exista suporte a SLP. SLP centraliza as informações em dire-

tórios, isto não deveria ser uma limitação, mas além das informações as requisições também são realizadas por esses diretórios.

No Brasil apesar de termos poucas equipes realizando pesquisas nessa área, já temos alguns trabalhos publicados como o DSB, ou mesmo trabalhos envolvendo semântica. No SBCUP de 2010 foi publicado um artigo falando sobre o "EXEHDA-SD" [GEYER 2010], o qual trata de semântica e computação ubíqua, porém descreve semanticamente recursos e não serviços, excluindo o trabalho das comparações realizadas nesse capítulo.

O pequeno número de trabalhos comparados se deve a progressiva especialização dos trabalhos na area. Cada vez mais os modelos são aplicados a um domínio específico, a fim de obter o melhor desempenho e uma aplicabillidade mais concreta em seu domínio específico. Bons exemplos são: [Terziyan, Kaykova, Zhoytobryukh, 2010] onde um modelo é criado para resolver problemas provenientes da criação de estradas inteligentes; outro exemplo é [Raafat e Cecelja, 2011] onde um modelo ontológico é idealizado para lidar com serviços móveis em um ambiente de telemedicina.

4 PROPOSTA DO MODELO

Neste capítulo é apresentado o modelo proposto, como ele funciona e como ele contribui para a criação de um ambiente ubíquo.

4.1 VISÃO GERAL

A estrutura básica do modelo como pode ser vista na Figura 9, é totalmente baseada na arquitetura SOA, com diferenças nas descrições de serviços e nas buscas, além da dinâmica de invocação também ter sido aprimorada. Estes aspectos são melhor detalhados ao longe deste capítulo.

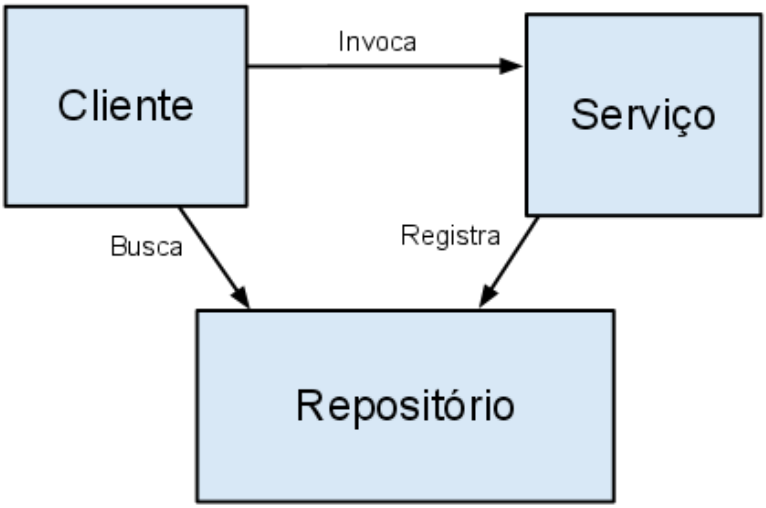

Figura 9 – Arquitetura do modelo

A infra-estrutura do modelo visa prover um ambiente onde dispositivos móveis, mas não limitado a eles, possam prover e invocar serviços, e que esses serviços possam ser dinamicamente descobertos por outros dispositivos. O diferencial do presente trabalho está na descoberta desses serviços utilizando semântica, realizada mesmo por dispositivos com recursos limitados. Um modelo de descoberta através do nome do serviço ou um identificador não é satisfatória

para que esse modelo se torne viável no dia-a-dia das pessoas. Para a descoberta de serviços atingir um alto grau de utilização é necessário que esses serviços sejam descritos de maneira que seu significado possa ser compreendido em todos os contextos. Uma palavra ou uma frase pode conter diferentes significados, tornando a busca imprecisa.

Serviços usualmente são sintaticamente descritos, e essas descrições possuem as informações necessárias para a invocação de um serviço, como protocolos, métodos, e parâmetros. Contudo, um dispositivo que encontra esses serviços, mesmo tendo as informações de como realizar a invocação, não tem como saber o que efetivamente esse serviço faz.

No modelo proposto descrevemos esses serviços de maneira semântica, utilizando ontologias. Ontologias permitem a criação e o compartilhamento de conceitos e o relacionamento entre eles. Descrevendo os serviços semanticamente, de maneira que dispositivos computacionais possam entender essa descrição, podemos minimizar a intervenção humana na comunicação entre serviços. Um cliente pode buscar um serviço por sua funcionalidade, dados de entrada e saída ou algum outro parâmetro determinante.

Processar a descrição semântica de um serviço ou uma busca semântica é uma tarefa que exige uma grande quantidade de recursos computacionais, como processamento e memória, muitas vezes insuficientes em dispositivos móveis ou com recursos limitados. A fim de sanar este problema, o modelo prevê repositórios semânticos presentes na rede, capazes de processar essas informações semânticas e responder a quem requisita qual serviço deve ser invocado. Os repositórios devem preferencialmente ser providos por dispositivos sem grandes restrições de recursos.

Portanto, o modelo é formado por serviços, que podem ser providos e acessados por qualquer dispositivo com algum recurso computacional e conectividade. Clientes que irão invocar tais serviços, da mesma maneira que os serviços, podem executar em diversos dispositivos. Por sua vez, o repositório formado por um serviço implementado seguindo os padrões do modelo, se comunica com uma *engine* de processamento semântico, a qual é acessada como um *Web Service* comum.

4.2 OS COMPONENTES QUE FORMAM O AMBIENTE

Como apresentado anteriormente, o ambiente de execução do modelo é formado por diversos dispositivos, que podem assumir alguns papéis diferentes. Na Figura 10 podemos observar a arquitetura SOA formada pelo projeto e os papéis que dispositivos podem assumir.

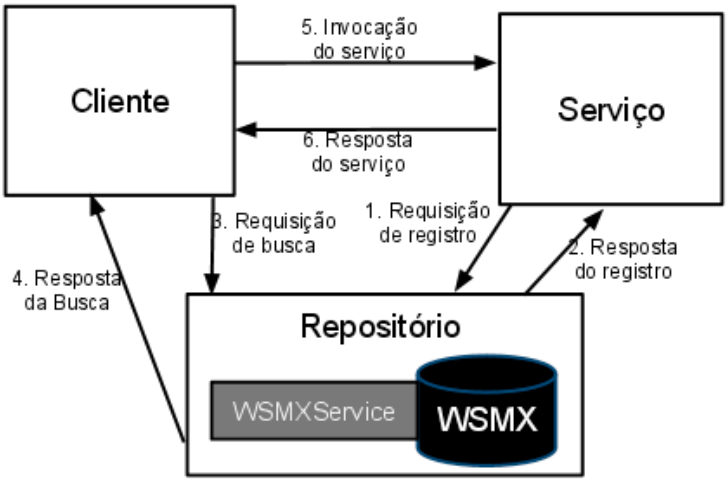

Figura 10 – Componentes do Modelo

O cliente é responsável por portar a descrição dos serviços semânticos desejados e invocá-los, a fim de realizar as ações necessárias para atingir um objetivo. A descrição dos serviços é representada na forma de um ou mais *goals*, um dos artefatos (*top entities)* do WSMO. Cada *goal* pode descrever diversos aspectos de um serviço desejado, como parâmetros de entrada e saída de um serviço, seus efeitos, entre outras características.

Um cliente pode ou não apresentar uma interface para interação humana, como uma aplicação rodando em um *smartphone*, um *tablet* ou mesmo um *desktop*, na qual o usuário expressa suas preferências para que o cliente acione os serviços necessários e execute algumas ações, ou mesmo um serviço que precisa invocar outro serviço, exercendo o papel de cliente nesse momento.

Mesmo os clientes que são aplicações com interface humana, tendem a não necessitar de interações constantes, visto que um perfil pode

ser cadastrado. Baseando-se em informações de contexto e nesse perfil, a aplicação pode invocar serviços de maneira transparente ao usuário para adaptar o ambiente conforme seu desejo previamente expressado.

Um cliente pode conter *goals* pré-cadastrados, os quais o usuário pode escolher e configurar parâmetros conforme suas preferências. Podem ser utilizados *goals* disponibilizados na Web, sendo acessados através de sua IRI, ou mesmo de uma maneira mais avançada, porém não tão amigável, apresentando um campo de texto para que o usuário crie seus próprios *goals*. Como aqui apresentamos apenas um modelo, o desenvolvedor que cria um cliente deve decidir qual estratégia utilizar para melhor atender suas necessidades, sendo que não há obrigatoriedade de se limitar às possibilidades aqui listadas.

Um cliente é basicamente uma aplicação cliente que implementa o DPWS, a única especificidade é que ela deve seguir um fluxo de invocação determinado quando se conecta a rede para encontrar o WSMXService e a busca deve ser realizada por este serviço caso ele seja encontrado.

Serviços são as entidades responsáveis pela execução das tarefas. Um serviço possui operações, as quais são responsáveis pela realização de tarefas específicas. Serviços são representados por outro dos artefatos do WSMO. A descrição semântica criada desses serviços é baseada em ontologias que definem conceitos e relações. Essa descrição deve conter todas as informações de um serviço que podem ser úteis para que possa ser encontrado utilizando um *goal*.

Serviços podem realizar qualquer tarefa viável computacionalmente, desde a invocação de outro serviço ou execução de um comando, até rotinas complexas.

O modelo aqui apresentado visa que a maior parte possível de dispositivos proveja suas funcionalidades através de serviços, que sejam passíveis de uma descoberta automatizada. Qualquer dispositivo com algum recurso computacional e conectividade pode ser adequado para sua interoperabilidade com o modelo.

Um dispositivo pode prover diversos serviços distintos ao mesmo tempo. Cada serviço deve possuir pelo menos uma descrição semântica. Clientes muitas vezes serão executados em dispositivos com recursos limitados. Para evitar o processamento semântico nesses dispositivos, economizando tempo do usuário e se for o caso bateria do dispositivo, sugerimos que cada operação de um serviço possua sua própria descrição, assim um cliente já sabe exatamente qual dispositivo, serviço e operação deve ser invocada sem a necessidade de um grande processamento local.

Um serviço implementa o padrão DPWS, quando entra na rede executa o fluxo de invocações para encontrar o WSMXService e se registrar no servidor. O *namespace* da descrição semântica do serviço deve seguir o padrão "url/nomedoservico#operação", assim o cliente pode facilmente identificar o serviço e a operação a ser invocada sem necessidade de realizar processamento semântico. A maior diferença desse serviço para um serviço DPWS comum será a criação da descrição semântica.

O repositório é responsável por armazenar as entidades WSMO, incluindo serviços e *goals*, além de realizar o processamento semântico necessário para verificar a compatibilidade de um *goal* com um serviço. Por conta da necessidade de recursos computacionais muitas vezes indisponível em dispositivos com recursos limitados e certa estabilidade na conexão, aconselhamos que este componente seja mantido em um dispositivo sem limitação de recursos e que permaneça o máximo de tempo possível conectado na mesma rede.

O repositório é dividido em componentes, os quais serão detalhados separadamente. São dois principais componentes: WSMX e WSMXService, os quais ainda podem ser subdivididos em componentes.

WSMX é o ambiente de execução semântico do WSMO e realiza o processamento, suas funções são acessadas através de *Web Services*, e sua linguagem nativa é o WSML. Uma descrição mais detalhada de seus componentes internos pode ser encontrada na sessão 2.2.2.4.

O *WSMX Service* foi totalmente implementado para o adequado funcionamento do modelo aqui proposto, é dividido em alguns componentes bastante simples e com funções muito específicas, que podem ser visualizados na Figura 11. A DPWS Interface implementa o padrão DPWS e é responsável por receber as requisições dos clientes e serviços. As requisições são repassadas para a WSMX Interface, a qual possui uma implementação para converter as requisições para WSML quando necessário e invocar os serviços do WSMX, a invocação é realizada através de operações que utilizam as funções da WSMX API. A WSMX API é uma biblioteca fornecida pelos desenvolvedores do WSMX para integração de sistemas com o mesmo. O verificador de disponibilidade de serviço foi implementado para funcionar como um componente de *Ranking* que executa de tempos em tempos uma operação para verificar se os serviços armazenados no WSMX estão disponíveis. Em caso positivo, a credibilidade do serviço é incrementada, e em caso negativo é decrementada. Quando a operação realizada no WSMX é uma consulta, e mais de um serviço é retornado, antes da resposta ser devolvida ao

cliente, a lista de serviços é reordenada conforme a credibilidade dos serviços através do ordenador de buscas que utiliza o *ranking* criado pelo verificador de disponibilidade.

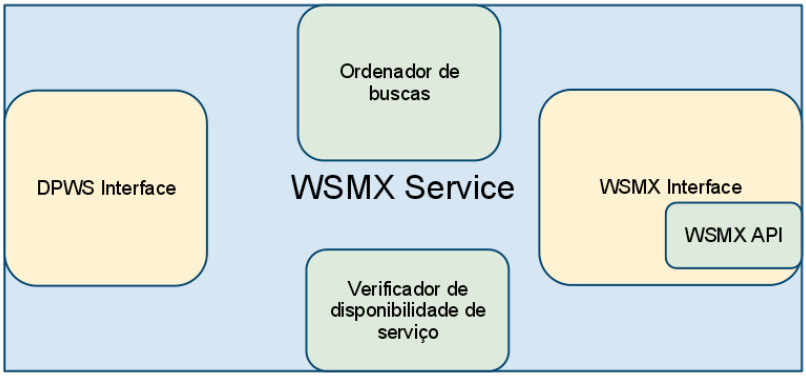

Figura 11 – Arquitetura do WSMX Service

O WSMX Service possui quatro operações, as quais estão encapsuladas pela DPWS Interface:

- Registrar Ontologia - que é utilizada para armazenar as ontologias criadas no repositório para que sejam utilizadas no processamento semântico de serviços;
- Registrar Serviço - esta operação deve ser invocada por todos os serviços para que estes sejam registrados e possam ser encontrados posteriormente;
- Registrar Goal - permite que goals sejam armazenados para serem executados posteriormente, evitando assim que sempre que um cliente precise encontrar um serviço seja necessário enviar um goal. Apenas a identificação do mesmo pode ser enviada para que o WSMX o execute.
- Executar Goal - esta operação é a principal do componente, visto que é através desta operação que a ordem para buscar certo serviço é enviada. Porém, sem as duas primeiras operações seria inviável manter um sistema com tal dinâmica.

O WSMX possui em sua interface diversas operações, apenas uma pequena parte dessas operações é utilizada pelo WSMX Service. A WSMX Interface invoca as operações correspondentes as operações do DPWS Interface, além das quatro anteriormente citadas, ainda a operação *retrieveWebServices* é utilizada para a verificação de disponibilida-

de dos serviços armazenados no WSMX, possibilitando que o ordenador de buscas funcione.

A interface do WSMX possui operações para o registro de ontologias tanto por sua descrição quanto pela sua IRI. O mesmo ocorre para cada uma das outras entidades do WSMO: *service*, *goal* e *mediator*. Além do registro, todas as entidades possuem interfaces para exclusão, visualização e listagem. As operações para execução de *goals* são três: uma recebe como parâmetro o *goal* e pesquisa em toda a base de dados do WSMX; outra recebe o *goal* e uma ontologia com os conceitos utilizados no *goal*; e a última recebe como parâmetro o *goal* e um conjunto de instâncias, no qual o WSMX verifica quais instâncias são compatíveis com o *goal*. O restante das operações contidas não é de grande relevância para o protótipo aqui apresentado.

O sistema de *ranking* é extremamente simples. Uma função é invocada repetidamente com em intervalos de tempo pré-determinados. Esta operação requisita a lista atual de serviços armazenados no repositório, de posse da lista os serviços são verificados individualmente, através de uma invocação, para checar se estão online. A cada verificação o sistema verifica se o serviço está na tabela de *ranking*, em caso negativo o serviço é cadastrado e a operação segue, caso o serviço esteja funcionando o sistema incrementa o contador do serviço. Caso o serviço não responda, esse contador é decrementado. Todos os serviços iniciam com o contador no 0. Quanto maior o valor do contador de um serviço, maior sua confiabilidade.

O ordenador de buscas, quando invocado, apenas executa um algoritmo de ordenação nos serviços baseados no valor do contador da tabela de *ranking*, uma tarefa bastante simples, mas que pode poupar tempo, processamento e comunicação, que no caso de um dispositivo com recursos limitados pode ser de grande ajuda.

4.3 DINÂMICA DE EXECUÇÃO

Na sessão anterior foram apresentados os componentes do modelo proposto no presente trabalho e o que eles fazem. Essa sessão visa esclarecer o funcionamento de cada componente apresentando e como eles interagem para o funcionamento do modelo.

São executadas basicamente 3 diferentes macro operações: Entrada de um serviço na rede, entrada de um cliente na rede e a busca por um serviço. A invocação de um serviço não é abordada nessa sessão,

pois posteriormente à descoberta de um serviço, toda a invocação segue exatamente o funcionamento do modelo proposto no padrão DPWS, apresentado na sessão 2.1.5.

4.3.1 Entrada de um serviço na rede

Ao se conectar a uma rede, é preciso que um serviço viabilize sua descoberta. Esse processo exige o registro deste serviço em um repositório, para que sua descrição semântica pode ser consultada a cada *goal* executado. Para realizar seu registro em um repositório, o primeiro passo deve ser a descoberta do serviço que representa o repositório.

Todos os serviços que realizam a comunicação com o repositório (WSMX) devem ser serviços DPWS, utilizando como padrão na nomenclatura deste serviço a identificação "WSMXService". Quando um serviço entra na rede, ele envia uma requisição *Probe* do DPWS buscando por todos os serviços identificados por esse nome. Cada resposta à requisição representa um repositório presente na rede. Com a informação dos repositórios presentes, o serviço pode realizar seu registro em um ou mais repositórios, de acordo com a vontade do programador. Quanto maior o número de repositórios, mais fácil um serviço ser encontrado. Porém, o número de requisições de registro aumenta. Considerando um dispositivo que use bateria, cada requisição exige certa quantidade de energia, que pode variar de acordo com o modelo. Contudo podemos afirmar que em uma rede onde são encontrados cinco repositórios ao se registrar em apenas um deles, estamos gastando apenas 20% da bateria do que gastaríamos se registrando nos cinco.

Para realizar o registro seguindo o fluxo normal do DPWS, o serviço realiza uma invocação no *WSMX Service* passando como parâmetro sua descrição semântica, deixando a cargo do *WSMX Service* se comunicar com o WSMX e persistir tal descrição. Todo esse processo é ilustrado na Figura 12.

A descrição semântica de um serviço é representada pelo elemento *Web Services* do WSMO, que pode descrever cada aspecto de um serviço e suas operações. Como descrito anteriormente, para minimizar o processamento realizado nas aplicações clientes é sugerido que cada descrição referencie apenas uma das operações do serviço.

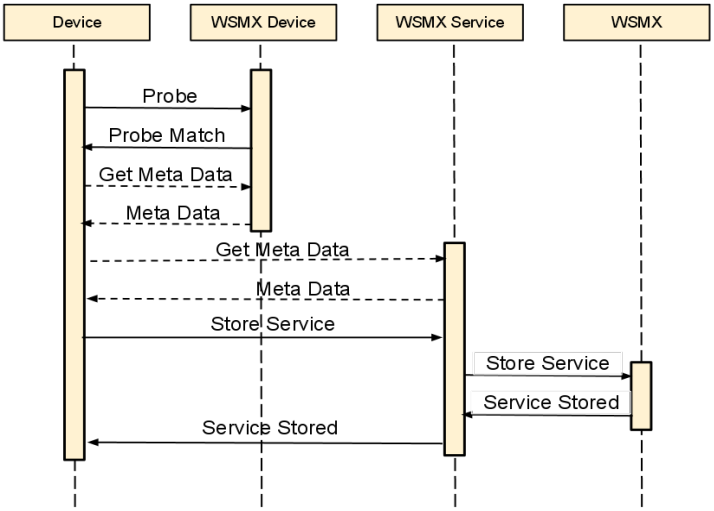

Figura 12 – Entrada de um Serviço na rede

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.inf.ufsc.br/~rbesen/Service/LightService#",
     discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#",
     cE _"http://www.inf.ufsc.br/~rbesen/Ontology/ContextOntology#" ,
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#"}

webService AdjustLight

     importsOntology
           _"htttp://www.inf.ufsc.br/~rbesen/Ontology/ContextEntity"

capability LightServiceCapability
     nonFunctionalProperties
          discovery#discoveryStrategy hasValue discovery#LightweightRuleDiscovery
     endNonFunctionalProperties

postcondition serviceLightStatusCapability
    definedBy
          ?light memberOf cE#Light[
               cE#location hasValue ?location
          ] and ?location memberOf cE#OfficeRoom.

interface LightServiceInterface
     choreography LightServiceChoreography
     stateSignature LightServicesignature
          importsOntology
            _"http://www.inf.ufsc.br/~rbesen/Ontology/ContextOntology#"
          in
               concept cE#Location
          out
               concept cE#Light
```

Figura 13 – Exemplo de descrição

Na Figura 13 temos um exemplo da descrição semântica de um serviço. Essa é uma descrição bastante simples, onde na "*wsmlVariant*" é descrita a variante do WSMO utilizada para descrição do serviço. Em "*namespace*" temos primeiro o *namespace* do serviço, depois referenciamos outros artefatos WSMO que serão utilizados através de seus *namespaces*. Após esse cabeçalho temos a descrição do serviço, onde temos o nome, as ontologias importadas e a estratégia de descoberta a ser utilizada pelo *reasoner* semântico. O bloco "*postcondition*" é a descrição do serviço utilizando os conceitos definidos em ontologias. Os parâmetros de entrada e saída do serviço são definidos no bloco *interface*, utilizando também conceitos definidos em ontologias.

4.3.2 Entrada de um cliente na rede

O processo inicial da entrada de um cliente na rede é exatamente o mesmo da entrada de um serviço, ou seja, encontrar o repositório. Porém, a localização do repositório não ocorre para que o cliente possa se registrar, mas sim para ter conhecimento de onde procurar por um serviço quando necessário.

Com o conhecimento da localização dos repositórios presentes na rede, um cliente pode realizar a busca em um ou mais repositórios. Como no serviço isso depende da estratégia utilizada no sistema. Um cliente pode enviar requisições para diversos repositórios ao mesmo tempo, dessa maneira possivelmente economizaria algum tempo na busca por serviços. Porém, provavelmente seriam realizadas requisições desnecessárias desperdiçando bateria de um dispositivo se for o caso. Pode-se realizar a busca em um repositório por vez, ou usar algum algoritmo de seleção para escolher em qual repositório buscar, mas isso depende da natureza da aplicação.

A Figura 14 apresenta o fluxo de requisições da entrada de um cliente na rede.

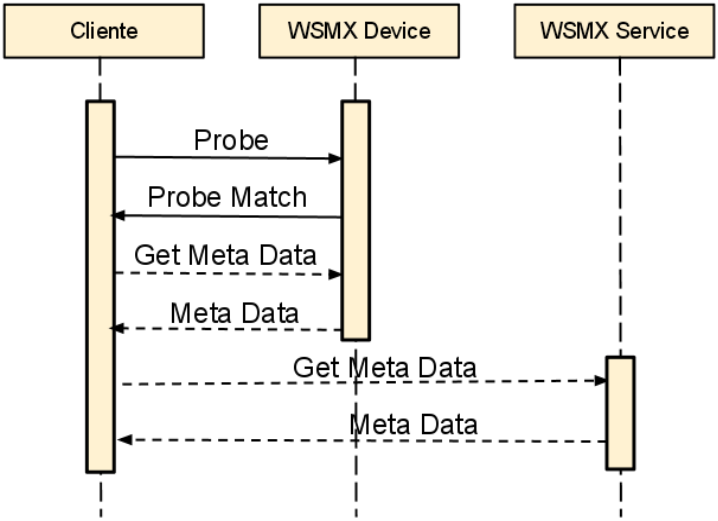

Figura 14 – Entrada de um cliente na rede

4.3.3 Busca por um serviço

Clientes realizam a busca no repositório através de *goals,* que devem descrever o serviço desejado. Podem ser descritas características como: efeito de uma requisição no serviço, pré e pós condições de invocação, parâmetros de entrada e saída de uma operação, entre outros. Para detalhes da criação de *goals* é recomendada a leitura de [Fensel at. al. 2007].

A Figura 15 ilustra a estrutura de um *goal*, na qual é expressada uma busca bastante simples baseada em conceitos descritos em uma ontologia. Os cabeçalhos "*wsmlVariant*" e "*namespace*" funcionam exatamente da mesma maneira que na descrição de serviços. O restante é bastante semelhante: temos o nome do *goal,* seguido das ontologias importadas e posteriormente a descrição da busca, utilizando os conceitos da mesma ontologia utilizada na descrição do serviço da Figura 13.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.inf.ufsc.br/~rbesen/Service/LightGoal",
     cE _"http://www.inf.ufsc.br/~rbesen/Ontology/ContextOntology#" ,
     discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#" }

goal LightGoal

     importsOntology
           _"htttp://www.inf.ufsc.br/~rbesen/Ontology/ContextEntity"

capability goalCapability
    postcondition
       definedBy
            ?light memberOf cE#Light[cE#location hasValue ?location
                ] and ?location memberOf cE#OfficeRoom.
```

Figura 15 – Descrição de um goal

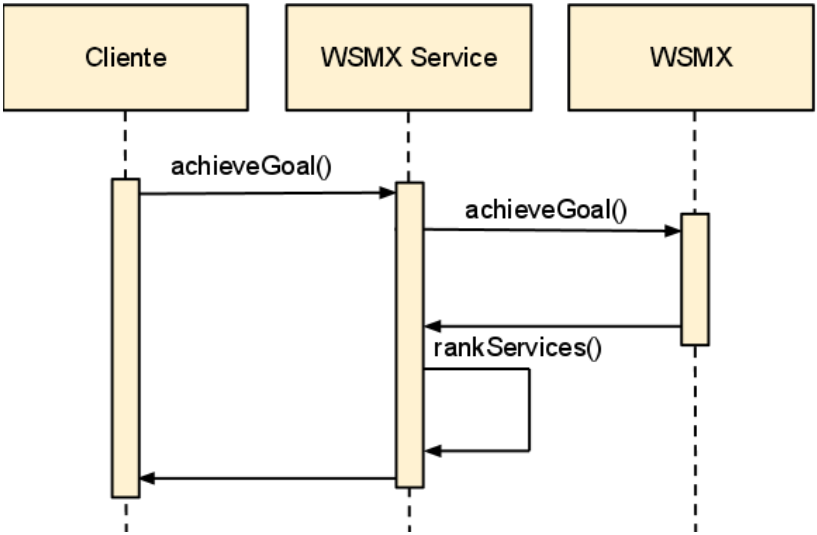

Figura 16 – Busca por um serviço

Para que a busca semântica funcione, o cliente precisa ter a informação de onde realizar essa busca. A sessão anterior descreve como obter essa informação. O cliente inicia a busca a partir de algum evento, como uma requisição do usuário ou uma mudança no contexto, dependendo da aplicação utilizada. Como conseqüência do evento, o cliente invoca o serviço *achieveGoal* no *WSMX Service* passando como parâmetro o *goal* com as características do serviço desejado. O *WSMX Service* estrutura esse parâmetro de maneira que o WSMX possa compreender essa informação, e posteriormente a invocação da operação no

repositório é realizada. Caso sejam encontrados serviços compatíveis, esses farão parte do retorno da operação. Caso seja retornada uma lista de serviços, o *WSMX Service* realiza a ordenação destes resultados conforme sua disponibilidade. A lista ordenada é encaminhada ao cliente, que dependendo de sua política de invocação invocará um ou mais serviços. O fluxo realizado para execução desta tarefa pode ser visualizado na Figura 16.

O processo de invocação de um serviço segue o padrão normal de uma requisição DPWS. A sessão 2.1.5 descreve esse processo mais detalhadamente.

4.4 CONSIDERAÇÕES

Neste capítulo foi apresentado modelo proposto nessa dissertação e seu funcionamento. A arquitetura é basicamente a mesma de qualquer outro modelo SOA, mantendo um baixo nível de acoplamento. Contudo pequenas diferenças em seus componentes possibilitam a adição de funcionalidades chave para a melhoria do processo de descoberta de serviços.

Serviços semanticamente descritos possibilitam que sua finalidade seja compreensível por máquinas otimizando a descoberta e a composição de serviços. A capacidade de se registrar em um repositório automaticamente, através da descoberta do repositórios e pro-atividade do serviço eliminam a necessidade de intervenção humana nessa etapa.

A possibilidade do componente do sistema seja cliente ou serviço, descobrir um repositório conectado a rede, elimina o problema existente na arquitetura SOA convencional, onde é necessário conhecer a localização do repositório de serviços. Além do que mesmo conhecendo a localização do repositório nada indica que o serviço foi registrado.

Ao entrar na rede um cliente mesmo que não realize uma invocação imediata busca pelos repositórios disponíveis. Uma lista de repositórios é mantida pelo cliente, evitando que no momento da busca seja necessário buscar por um repositório, o que poderia aumentar o tempo de resposta.

5 AMBIENTE E RESULTADOS EXPERIMENTAIS

Para verificação do modelo proposto, criamos um caso de uso fictício onde aplicamos os conceitos e padrões apresentados nesse trabalho. Os testes têm como objetivo não apenas mostrar o funcionamento do modelo, mas também capturar os tempos para descoberta e acesso de serviços para verificar a viabilidade do modelo em um ambiente real.

Neste capítulo primeiro será apresentado o caso de uso utilizado, será apresentada a ontologia criada para representar os conceitos e ontologias utilizadas para descrever os serviços e *goals*. Posteriormente a configuração do ambiente onde os testes foram executados será detalhada, e por fim serão apresentados os resultados obtidos.

5.1 AMBIENTE DE TESTES

O ambiente criado para realizar os testes é composto por diversos componentes. O WSMX, versão 0.5, foi executado um uma máquina virtual com Windows XP com 2Gb de memória RAM e 1 CPU, na qual foram armazenados inicialmente 25 ontologias, 40 serviços e 36 *goals*. O WSMXService e a aplicação cliente foram executadas em um *notebook* com 4Gb de memória RAM e processador Intel Core2Duo 2.4GHz executando sistema operacional MacOSX *Snow Leopard*. Outros serviços, inclusive o serviço desejado, chamado *LightService*, foram executados em um *notebook* com 3Gb de memória RAM e processador Intel Core2Duo 2.26GHz executando sistema operacional Ubuntu 10.10. Os dispositivos móveis foram emulados nos *notebooks* utilizando recursos da biblioteca WS4D. Toda a comunicação foi realizada via rede *wireless* IEEE 802.11g.

Cada requisição foi repetida por pelo menos cem vezes e o resultado obtido é uma média destas requisições. Posteriormente o número de serviços, *goals* e ontologias armazenados no WSMX foi aumentado para verificar seu comportamento.

5.2 CASO DE USO

A aplicação mais evidente do modelo neste trabalho apresentado se dá em um ambiente de automação de casas, prédios, escritórios e lugares em geral, conhecidos como *smartspaces*. Também temos diversas aplicações na indústria, ou mesmo em sistemas automotivos, mas *smartspaces* foram escolhidos por consistirem em uma aplicação mais impactante no dia-a-dia das pessoas.

Para melhor ilustrar o caso de uso, será utilizada uma história descrevendo os aspectos imaginados para criar o ambiente em que aplicamos o modelo. Fulano é analista de sistemas em uma grande empresa de tecnologia, e é um aficionado em tecnologia, o que é bastante comum dada sua profissão. Quando comprou seu apartamento queria automatizar tudo que fosse possível, o que nos dias atuais exigiria uma pequena fortuna. Depois de muita discussão com sua esposa, Fulano finalmente conseguiu permissão para adquirir e instalar todos os dispositivos necessários. *Dimmers*, lâmpadas, controladores de cortina, fechaduras, dispositivos de mídia, tudo que encontrou no mercado. Porém, mesmo depois de tudo instalado e funcionando, Fulano encontrou algumas dificuldades, pois tudo era controlado separadamente. Então, apesar de ser tudo automatizado, ele tinha praticamente o mesmo trabalho que teria para operar tudo manualmente.

Insatisfeito com a situação, Fulano resolveu buscar uma solução não apenas para integrar seus dispositivos, mas de alguma forma melhorar sua experiência em casa. Depois de muito pesquisar na internet e perguntar a seus colegas de trabalho, encontrou este trabalho. Fulano gostou do trabalho e viu que todas as tecnologias eram abertas e seguiam padrões de mercado, então pensou, "- Ótimo, não precisarei gastar mais dinheiro e evitarei mais uma discussão em casa". Fulano baixou os softwares necessários, leu a documentação e começou a implantação em sua casa. Visto que era analista de sistemas e tinha algum conhecimento de programação, não teve muitas dificuldades.

Fulano contou a seus colegas de trabalho sobre sua descoberta, nem precisou pedir para que o ajudassem, então rapidamente implantaram o repositório e depois de algumas noites em claro já haviam criado serviços para todos os dispositivos. A única parte que faltava era criar o cliente para que pudesse se comunicar com esses serviços. Desenvolver o cliente seria extremamente simples, porém definir quais *goals* este devia possuir para que tudo funcionasse de maneira integrada era o maior desafio. Mas Fulano sabia exatamente o que queria e iniciou a criação

de seus *goals*, que foram sendo criados com a possibilidade de ajustar parâmetros através da interface do cliente. Fulano colocou suas preferências como parâmetros *default*.

Depois de tudo implantado e testado, Fulano está ansioso para testar sua infraestrutura de descoberta semântica de serviços. No seu horário de almoço no trabalho, Fulano olha a grade de programação da televisão e encontra um filme que deseja assistir quando chegar em casa. Utilizando seu *smartphone* ele sinaliza que em certo horário deseja assistir televisão em um determinado canal.

Ao chegar em casa, baseado em informações sobre sua localização, o aplicativo em seu *smartphone* identifica que o usuário está em casa. Próximo do horário do programa, um alarme é gerado para avisar que o item agendado terá início. Assim que o usuário confirma que manterá a programação, todos os serviços que podem interferir na qualidade da experiência do usuário são buscados e ajustados conforme suas preferências.

Baseado na ontologia criada para descrever os dispositivos e o ambiente, a aplicação cliente busca todos os fatores ambientais que influenciam os dispositivos Televisão e do *Home Theater*, identificando imagem e som. Dispositivos que controlam janelas, cortinas e lâmpadas são identificados como fatores que influenciam diretamente na imagem e som, e estes dispositivos configuram o ambiente para as preferências do usuário para a ocasião específica. A Televisão é ativada no canal desejado e o volume do *Home Theater* é ajustado.

5.3 CRIAÇÃO DOS TESTES

Para validar o modelo criado, foi realizada uma simulação de um ambiente semelhante ao idealizado no caso de uso descrito, onde foi desenvolvida uma ontologia e foram descritos alguns serviços e *goals* que a utilizam.

Nas Figuras 17 e 18 é apresentada a ontologia criada, que foi dividida em duas partes para melhor visualização. Na Figura 17 podemos visualizar a parte da ontologia onde são descritos recursos relativos a localização geográfica - informações bastante úteis quando se deseja criar um contexto para usuários de dispositivos móveis, ou mesmo para os serviços. A Figura 18 descreve dispositivos e uma relação de quais fatores ambientais estes influenciam ou são influenciados. Unindo os conceitos apresentados nas duas figuras, podemos descrever um serviço

expressando, além de sua funcionalidade, sua localização, como ele influencia o ambiente em que se encontra e como é influenciado por ele.

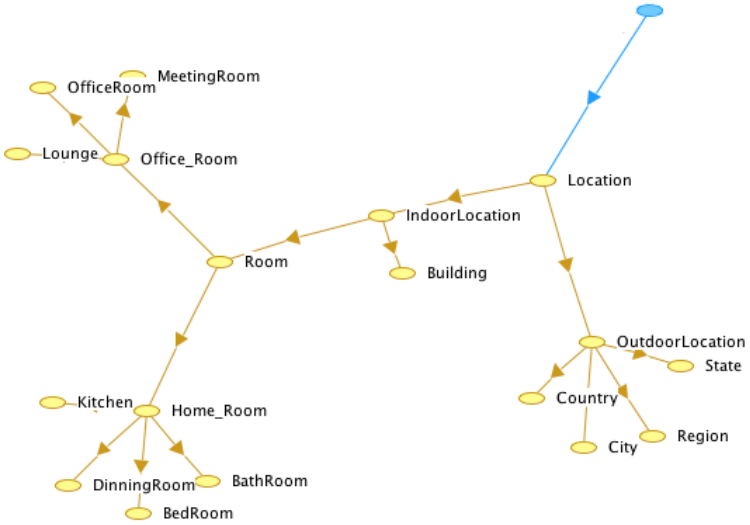

Figura 17 - Ontologia Parte 1

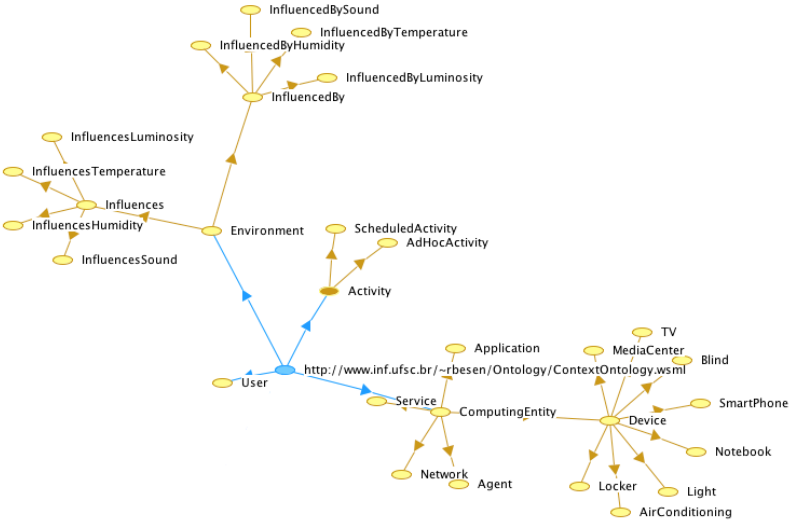

Figura 18 - Ontologia Parte 2

Utilizando a ontologia apresentada foram criados serviços para simular o controle de luzes e cortinas. Um serviço é semanticamente definido por suas características e ambos os serviços citados são extremamente simples. A seguir pode ser visualizado como o serviço que controla uma lâmpada é representado.

"serviceLightStatusCapability
definedBy
?light memberOf Ontology#Light[
Ontology#location hasValue ?location,
Ontology#InfluencesLuminosity hasValue 'true'
] and ?location = Ontology#Office1."

A descrição desse serviço nos diz que ele pertence ao conceito *Light* apresentado na ontologia, possui uma localização que é representada pelo conceito *location* contido na ontologia e influencia diretamente na iluminação do local, característica representada pelo conceito *InfluencesLuminosity* seguido do valor *true*. Na última linha é especificado que o local em que o serviço se encontra é o *Office1*, instância do conceito *OfficeRoom*. Este é um serviço bastante simples, porém se não fosse semanticamente descrito, seria mais complexo de maneira genérica identificar tal serviço baseando-se nas necessidades apresentadas por outro, como foi apresentado na sessão 5.1.

Com descrições como a apresentada acima, facilmente podemos criar uma rede de serviços que alteram uma característica em um ambiente, sem que a aplicação cliente ao menos conheça essa característica previamente. No caso de uso, a aplicação verifica que a Televisão pode ser influenciada pela iluminação e automaticamente busca todos os dispositivos que influenciam esse aspecto do ambiente.

A aplicação cliente encontra o serviço desejado utilizando descrições semânticas do serviço desejado. Para encontrar o serviço utilizado como exemplo, um *goal* bastante simples é utilizado. O *goal* só precisa especificar o conceito o qual busca, e propriedades do conceito podem ser utilizadas para refinar a busca. Para buscar todos os serviços que são representados pelo conceito *Light* apresentado na ontologia, e que estejam localizados em determinado lugar. Especificado por uma instância do conceito *OfficeRomm*, mais precisamente a instância *Office1,* pode ser utilizada a seguinte expressão:

```
"goalCapability
    definedBy
        ?light memberOf Ontology#Light[
            Ontology#location hasValue ?location]
    and ?location memberOf Ontology#Office1."
```

A idéia dos testes é que uma aplicação cliente entre na rede, encontre o WSMX Service, realize uma busca por serviços utilizando um *goal* e faça a invocação do serviço encontrado, fazendo com que todas as principais operações sejam executadas. Além de observar se o modelo é apto para concluir todas as etapas da invocação, também será obtido e observado o tempo de cada etapa, para verificar se seria viável em um ambiente real.

5.4 RESULTADOS EXPERIMENTAIS

Os testes realizados visaram medir o tempo de descoberta do WSMXService, o tempo de invocação do WSMXService, o tempo de invocação e processamento semântico do WSMX, o tempo de busca do serviço retornado e o tempo de invocação do mesmo. Nas medições foram observadas algumas questões importantes, principalmente referentes ao WSMX. Na primeira vez que um *goal* é executado, o tempo de resposta é bastante superior ao das execuções posteriores. Tal fato é justificado pela existência de uma *cache* no WSMX para diminuir o processamento necessário e acelerar o processo de busca. Na Figura 19 essa característica pode ser visualizada, no eixo X temos o tempo de resposta das invocações e no eixo Y, o passo que está sendo executado no momento.

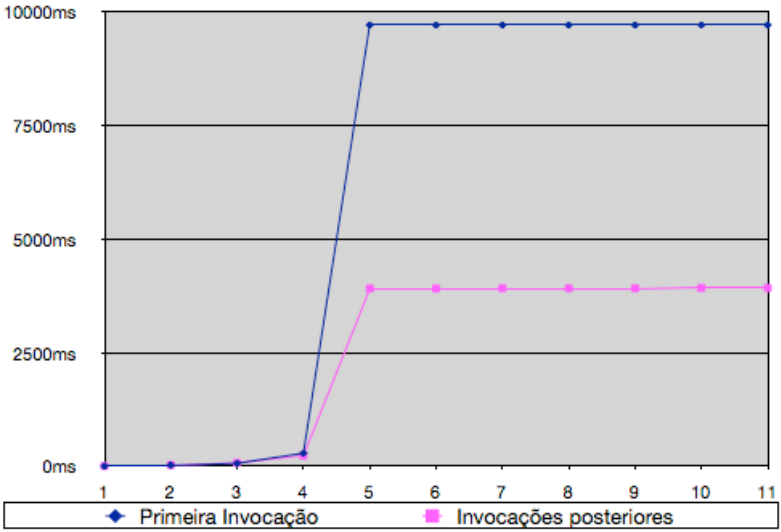

Figura 19 – Tempo de resposta do WSMX

Todo o processo de descoberta e invocação dos serviços foi dividido em 11 etapas:

1. ponto inicial e representa o início da busca pelo WSMX-Service;
2. momento em que o WSMXService é encontrado;
3. requisição DPWS é enviada para invocação do serviço;
4. invocação do WSMX;
5. WSMXService recebe a resposta do WSMX;
6. cliente recebe a resposta do WSMXService;
7. inicia a busca pelo "*LightService"*;
8. configurações do "*LightService*" são obtidas;
9. o serviço é invocado;
10. execução do serviço;
11. resposta é entregue ao cliente, encerrando a execução.

Como é observado na Figura 19, o tempo de execução do passo 4 é bastante superior a todo o restante do processo. Para uma melhor visualização, podemos dividir as etapas em duas grandes etapas: a primeira antes do processamento semântico, onde o cliente realiza a busca pelo "WSMXService", e a invocação de seu serviço para que o processamento semântico seja realizado; e a segunda, posterior ao processamento semântico, onde o cliente já obteve a informação do serviço que necessi-

ta, e faz a busca e invocação deste serviço. A primeira e segunda etapa são ilustradas respectivamente nas Figuras 20 e 21, onde da mesma maneira que a Figura 19 o eixo X representa o tempo de resposta das invocações e no eixo Y, o passo que está sendo executado no momento..

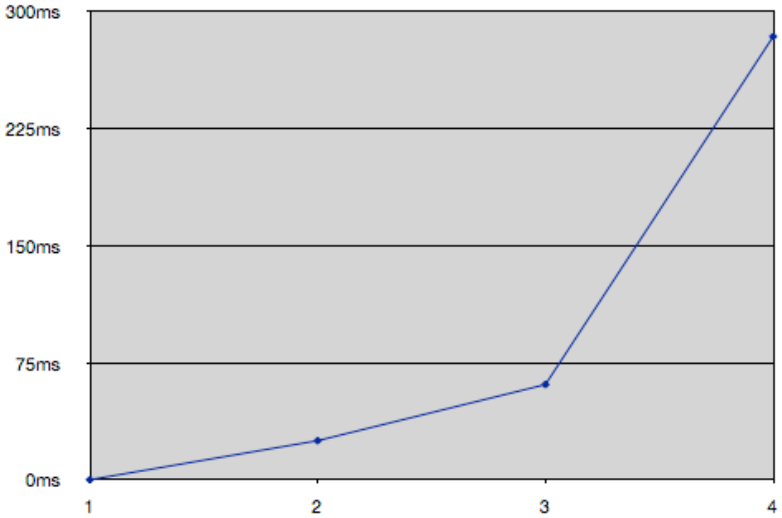

Figura 20- Primeira etapa de invocação

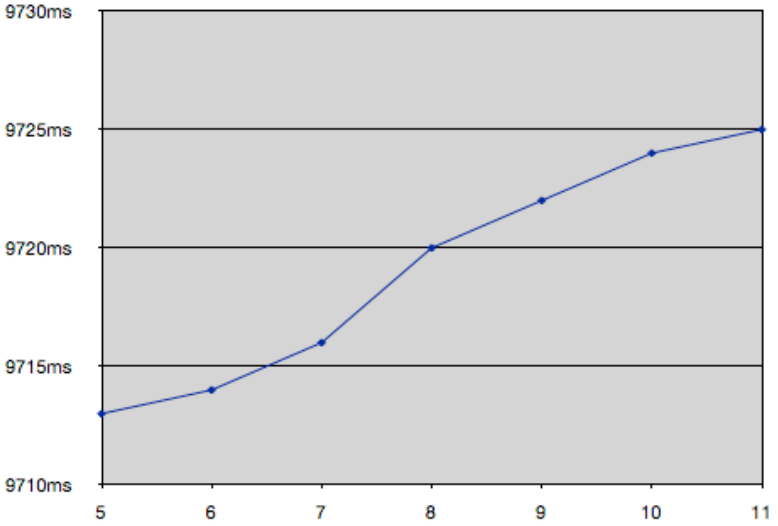

Figura 21 - Segunda etapa de invocação

Para tentar otimizar o tempo de processamento do WSMX, dobramos os recursos da máquina virtual, porém o tempo de processamento não apresentou alterações significativas. Por outro lado, ao aumentar o número de serviços na rede e no repositório para 100 serviços, apenas o tempo de processamento do WSMX na primeira invocação sofreu alteração, aumentando em 2 segundos. Analisando o comportamento do WSMX, pode ser observado que esse aumento se deve ao fato do sistema a verificar a compatibilidade de um *goal* com os serviços armazenados um a um. Contudo, nas invocações posteriores o tempo de invocação permaneceu o mesmo observado anteriormente.

5.5 ANALISE E COMPARAÇÃO DOS RESULTADOS

O modelo apresentado demonstra ser eficiente em todas as características apresentadas. Na comunicação entre serviços, Web Services apresentam a estabilidade e flexibilidade necessárias para a criação de um ambiente ubíquo. Também devido à adoção de Web Services, são eliminadas as restrições de linguagem e plataforma. O DPWS provê a infraestrutura necessária para que dispositivos, sejam eles com recursos limitados ou não, através de Web Services possam ser descobertos e acessados. O WSMO e seus componentes provêem a infraestrutura semântica para criação de ontologias e descrição dos serviços e *goals*.

Na Tabela 2 pode ser visualizada uma comparação entre os trabalhos relacionados e o modelo apresentado.

Tabela 2 - Comparação dos trabalhos relacionados com o modelo apresentado

Modelo	**Comunicação**	**Restriços de linguagem/ ambiente**	**Escopo**	**Suporte a Semântica**	**Suporte a Mobilidade**
AIDAS	WS	Java	qualquer um	Parcial	Limitado
HOME SOA	OSGi	Java	mídia residencial	Componentes externos	Total
DSB	WS	nenhuma	qualquer um	Nenhum	Total

P2P-Based Semantic Service	SLP	Suporte a SLP	Redes Ad Hoc	Total	Requisições centralizadas
Modelo Proposto	WS	nenhuma	qualquer um	Total	Total

5.6 CONSIDERAÇÕES

Os testes demonstram que o modelo é apto a concluir as tarefas que ele propõe. A expressividade da descrição de serviços, assim como a capacidade destes serem encontrados, depende diretamente da riqueza das ontologias criadas. O que tentamos apresentar nesse capítulo é que o modelo consegue atingir os objetivos, não é o foco medir a expressividade da ontologia utilizada como exemplo, com base nessas informações não achamos sentido em utilizarmos descrições de serviço mais complexas.

O tempo de resposta para a invocação parece um pouco alto, porém a pró-atividade do modelo deve mascarar tal atraso. Não se pode comparar o tempo de resposta aos outros modelos, pois estes não apresentavam nenhum resultado quantitativo.

O tempo levado pelas requisições para comunicação entre cliente e serviço e entre serviços se mantém no mesmo padrão do DPWS puro. Visto que não foram necessárias alterações nesse padrão, focando a maior parte da implementação do modelo no “WSMX Service“.

6 CONCLUSÕES E TRABALHOS FUTUROS

A presente dissertação atinge seus principais objetivos fornecendo um modelo de infra-estrutura para descoberta semântica de serviços em ambientes com dispositivos móveis. As características semânticas do WSMO, combinadas com a facilidade de comunicação do DPWS, possibilitam a criação de um ambiente muito semelhante ao descrito por Weiser no artigo em que conceitua um ambiente ubíquo [WEISER, 1993].

Se fosse necessário que o usuário esperasse 10 ou mais segundos por uma requisição, este seria um tempo bastante elevado. Porém, pela possibilidade do sistema ser pró-ativo, o usuário não efetua diretamente a requisição, apenas observa a ação sendo realizada. Ainda em um ambiente como apresentado no caso de uso deste trabalho, a tendência é que uma requisição se repita diariamente, fazendo com que em grande parte das requisições, o tempo de resposta seja bem inferior devido ao *cache* do WSMX.

Um possível aprimoramento do mecanismo proposto consiste na otimização do algoritmo de ordenação de serviços por disponibilidade, contudo este deve ser configurado de acordo com a aplicação do modelo. Em certos ambientes pode ser priorizada a confiabilidade e em outros a velocidade.

A principal questão a ser aprimorada é a diminuição do tempo de processamento semântico. Possivelmente a versão 1.0 do WSMX, que durante o desenvolvimento deste trabalho estava em versão beta, deverá melhorar esse tempo de resposta, pois teve seu código em grande parte reformulado. Esta versão beta não foi utilizada devido à falta de documentação e a problemas de estabilidade, mas deve trazer importantes aprimoramentos como a ordenação dos serviços localizados semanticamente, suporte a SAWSDL (padrão de WSDL anotado, recomendado para serviços web semânticos) e um componente para monitoramento dos serviços.

Como trabalhos futuros, além da otimização dos aspectos apresentados, citamos a possibilidade de criar uma interface comum onde poderiam ser acoplados adaptadores para múltiplos *reasoners*, possibilitando que o modelo suporte diversas modelos ontológicos. Assim, o trabalho não se limitaria apenas a WSMO.

O modelo aqui apresentado é apenas uma pequena contribuição para a criação de um ambiente semântico. A integração deste modelo

com outros projetos já desenvolvidos poderia nos dar um ambiente ainda mais próximo do idealizado por Weiser. A simples integração com o DSB, ao invés de utilizar diretamente o DPWS, nos daria a possibilidade de comunicação entre dispositivos que empregam protocolos de comunicação heterogêneos, como BlueTooth e RFID, por exemplo.

Apenas o suporte a semântica não é suficiente para que o ambiente se adapte ao usuário, a maneira como a semântica será utilizada e como o ambiente do usuário é mapeado são fundamentais para que o nível de interação e adaptação do ambiente seja satisfatório. A utilização de aplicações orientadas ao contexto deve enriquecer a experiência do usuário. As informações de contexto podem ser obtidas através de preferências do usuário, as quais podem ser obtidas através do preenchimento de formulários ou mesmo através do uso continuo dessas aplicações. Onde as informações referentes ao usuário são adaptadas conforme sua necessidade atual.

Contextos de usuário devem levar em consideração além das preferências pessoais, informações de localização e tempo. Visto que essas duas variáveis devem em grande parte das vezes influenciar diretamente na formação de contextos.

Poderiam ser trabalhadas ainda questões como segurança e endereçamento, o DPWS possui uma camada que lida com WS-Addressing, WS-Security e WS-Policy. Contudo, na criação do modelo e testes realizados estes aspectos foram ignorados por serem considerados fora do escopo da dissertação.

REFERÊNCIAS

ALONSO G., CASATI F., KUNO H., and MACHIRAJU V.. **Web Services. Concepts, Architectures and Applications**. Springer, 2004.

AYALA, D. et al. **Professional Open Source Web Services**. Wrox Press Ltd, 2002.

BAUMUNG, P. PENZ, S. KLEIN, M. **P2P-Based Semantic Service Management in Mobile Ad-hoc Networks**. Funded by the German Research Foundation within the SPP 1140. 2009.

BECHHOFER, S. et al. **OWL Web Ontology Language Reference**. W3C Recommendation. 2004. Disponível em http://www.w3.org/TR/owl-ref/

BELLAVISTA, P.; CORRADI, A.; MONTANARI, R.; STEFANELLI, C. **Context- aware middleware for resource management in the wireless Internet**, IEEE Transactions on Software Engineering 29 (12) (2003) 1086– 1099.

BERNERS-LEE, T.; HENDLER, J. and LASSILA. O. **The Semantic Web**. Scientific American, 284(5):34–43, May 2001.

BlueTooth Specification v. 3.0; April 2009, http://www.bluetooth.com/English/Technology/Building/Pages/Specification.aspx.

BOTTARO, A.; GÉRODOLLE, A. **HOME SOA**. Proceedings of the 5th international conference on Pervasive services, New York, USA, 2008.

BRAY, T. et al. **Extensible Markup Language 1.0(Fifth Edition)**. W3C Recomendation. 2008. Disponível em http://www.w3.org/TR/2008/REC-xml-20081126/

CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEERAWARANA, S. **WSDL specification**. W3C Recomendation. 2001. Disponível em http://www.w3.org/TR/wsdl.

COLOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos - conceitos e projetos**. Porto Alegre - RS: Artmed Editora, 2007.

DERI, **WSML version 1.0**, August 2008. Disponível em http://www.wsmo.org/wsml/wsml-syntax

DERI, **WSMT v0.1**, January 2005, Disponível em http://www.wsmo.org/TR/d9/d9.1/v0.1/20050127/

DERI**. WSMX version 0.5**, May 2008; Disponível em http://www.wsmx.org/

ERL T., **Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services**, Prentice Hall, 2004.

FALBO, R.A. et al. **Ontologias e Ambientes de Desenvolvimento de Software Semânticos**, Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, JIISIC'2004, Madrid, España, 2004.

FENSEL D. and BUSSLER C. **The Web Service Modeling Framework WSMF**. Electronic Commerce Research and Applications, 2002.

FENSEL, D. **Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition**. Springer, Berlin/Heidelberg, 2003.

FENSEL D. et al. **Enabling Semantic Web Services: The Web Service Modeling Ontology.** Springer, Alemanha, 2007.

FORMAN, George H and ZAHORJAN, J. **The chanllenges of mobile computing**. IEEE Computer Society Press Los Alamitos, CA, USA. 1994

GEYER, CLAUDIO F. D. at al. **Descoberta semântica de recurorsos na computaç!ao ubuíqua.** SBCUP 2010. Belo Horizonte, Brasil.

GRIMM, S.; KELLER, U.; LAUSEN H. and NAGYPAL G. **A Reasoning Framework for Rule-Based WSML**. In Proceedings of 4th European Semantic Web Conference (ESWC), Innsbruck, Austria, 2007

GRUBER T. R. **A translation approach to portable ontology specifications**. Knowledge Acquisition, 5:199–220, 1993.

GUDGIN, M. et al. **SOAP 1.2 specification**. W3C Recomendation. 2007. Disponível em http://www.w3.org/TR/soap12-part1/

KLEIN, M.; KONIG-RIES, B.; MUSSIG. M. **What is needed for semantic service descriptions - a proposal for suitable language constructs**. International Journal on Web and Grid Services (IJWGS), 1(2), 2005

KLYNE G. and CARROLL J. J. **Resource Description Framework (RDF): Concepts and Abstract Syntax.** W3C Recommendation, 2004. Available from http://www.w3.org/TR/rdf-concepts/.

MARTIN, D. et al. **OWL-S: Semantic Markup for Web Services**. 2004. Disponível em http://www.w3.org/Submission/OWL-S/#1

MARTIN, D. et al. **OWL-S 1.2 Release**. 2005. Disponível em http://www.ai.sri.com/daml/services/owl-s/1.2/

MCLLRAITH S., SON T. C. and ZENG H.. **Semantic Web Services**. IEEE Intelligent Systems, Special Issue on the Semantic Web, 2001.

MEDEIROS, G. ;SIQUEIRA, F. **DSB – Device Service Bus.** Proceedings of the 2009 ACM symposium on Applied Computing, New York, USA, 2009.

MICROSOFT Corporation. Devices Profile for Web Services (DPWS). 2006. Available at http://schemas.xmlsoap.org/ws/2006/02/devprof/.

Oracle. **J2ME 7 Specification**. 2011. Disponível em http://www.oracle.com/technetwork/java/javame/overview/index.html

OSGi ALLIANCE. **OSGi Specification 4.2**. 2010. Disponível em http://www.osgi.org/Specifications/HomePage

POLLERES, A.; BOLEY, H. KIFER, M. **RIF Datatypes and Built-Ins 1.0**. W3C Recommendation. 2010. Disponível em http://www.w3.org/standards/techs/rif#w3c_all

PRUD'HOMMEAUX, E. and SEABORNE, A. **SPARQL Query Language for RDF**. W3C Recommendation. 2008. Disponível em http://www.w3.org/standards/techs/sparql#w3c_all

RIEGEN C. V. e contribuidores. **UDDI Data Structure Specification**. 2002. Disponível em http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm

ROMAN D. et al. **Web Service Modeling Ontology**. Applied Ontology, 2005. Disponível em http://www.wsmo.org/TR/d2/v2.0/.

Salutation Consortium. **Salutation Architecture Specification**. 1999.

SATYANARAYANAN - **Pervarsive Computing - Vision and challenges**. IEEE Personal Communications, 2001

SINGH, Inderjeet et al. **Designing Web Services with the J2EE 1.4 Plataform JAX-RPC, SOAP, and XML Technologies**. United States: Addison- Wesley, 2004.

Sun Microsystems. **Jini Technology Core Platform Specification, v. 2.0.** June 2003. Disponível em www.sun.com/software/jini/specs/core2_0. pdf.

TERZIYAN, V.; KAYKOVA, O.; ZHOYTOBRYUKH, D. **UbiRoad: Semantic Middleware for Context-Aware Smart Road Environments**, *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference.* 2010. Disponível em http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5476731&isnumber=5476480

The Internet Engineering Task Force. **Service Location Protocol, version 2**. 1999. Disponível em http://www.ietf.org/rfc/rfc2608.txt

TONINELLI, A.; CORRADI, A.; MONTANARI, R. **AIDAS - Semantic-based discovery to support mobile context-aware service access**. Proceedings of Computer Communications, 2008.

UPNP Forum. **Arquitetura de Dispositivo UPnP v1.0**, 2006. Available at http://www.upnp.org/resources/documents.asp.

W3C. **Arquitetura de Serviço Web**. 2004. Disponível em http://www.w3.org/TR/ws-arch/.

W3C. **Especificação XML 1.0, quinta edição**. 2008. Available at http://www.w3.org/TR/2008/REC-xml-20081126/

WEISER, Mark. **Ubiquitous Computing.** IEEE Computer, 1993. 26(10):71– 72.